Safety Manual for S32V234

Devices Supported: S32V234, S32V232

Document Number: S32V234SM
Rev. 3, 10/2017

Contents

Section number **Title** **Page**

Chapter 1
Preface

1.1 Overview

This document discusses requirements for the integration and use of the S32V234 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 S32V234, 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 S32V234 is developed according to ISO 26262 and has an integrated safety concept.

1.2 Safety manual assumptions

During the development of the S32V234, 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 S32V234.

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 S32V234 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 S32V234. 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 S32V234 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 S32V234 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 26262-10:2011-2012 Annex A ISO 26262 and microcontrollers*.

1.5 Related documentation

The S32V234 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 S32V234 into safety-related systems, the following documentation will be available:

- Reference Manual (S32V234RM) - Describes the S32V234 functionality
- Data Sheet (S32V234DS) - Describes the S32V234 operating conditions
- Safety Manual (S32V234SM) - Describes the S32V234 safety concept and possible safety mechanisms (integrated in S32V234, 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)
- 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 S32V234 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 S32V234, the following information should be considered:

- The S32V234 is handled in accordance with JEDEC standards J-STD-020 and J-STD-033.
- The operating conditions given in the S32V234 Data Sheet.
- If applicable, any published S32V234 errata.

- The recommended production conditions given in the S32V234 quality agreement.
- The functional safety manager for the developed and deployed system is required to report all field failures of the S32V234 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 Target applications

As a SafeAssure solution, the S32V234 microcontroller targets applications requiring a high Automotive Safety Integrity Level (ASIL), especially:

- Front View
- Reverse View
- Surround View
- Data Fusion

2.2 Safety integrity level

The S32V234 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, of ISO 26262 or of IEC 61508). The S32V234 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 S32V234 is considered a Safety Element out of Context (SEooC) development.

The S32V234 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 S32V234, no specific safety function can be specified. Therefore, during the SEooC development of the S32V234, 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:

- **Surround view / Reverse view function (Application dependent)**: One or more camera pictures are shown on a screen. Continuous image update faults (frozen image display) require safety integrity of ASIL B. All other faults are perceived by the driver (no image, image distortion, color change, contrast change...).
- **Front View function (Application dependent)**: Software processing data from one or two cameras (ASIL B).
- **Data Fusion function (Application dependent)**: Software processing data from different sensors (ASIL B).
- **Data Fusion function (Application dependent)**: Software processing data from different sensors (ASIL C).

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

- **Input / Output Functions (Application dependent)**: Input / Output functions of the S32V234 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 S32V234 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 S32V234 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 S32V234 are:

- Operating correctly (see Figure 2-1 and section Correct operation)
- Explicitly indicating an internal error (indication on FCCU_F*n*, Figure 2-1)
- In reset (see Figure 2-1)
- Completely unpowered (see Figure 2-1)
- No output (see Figure 2-1)

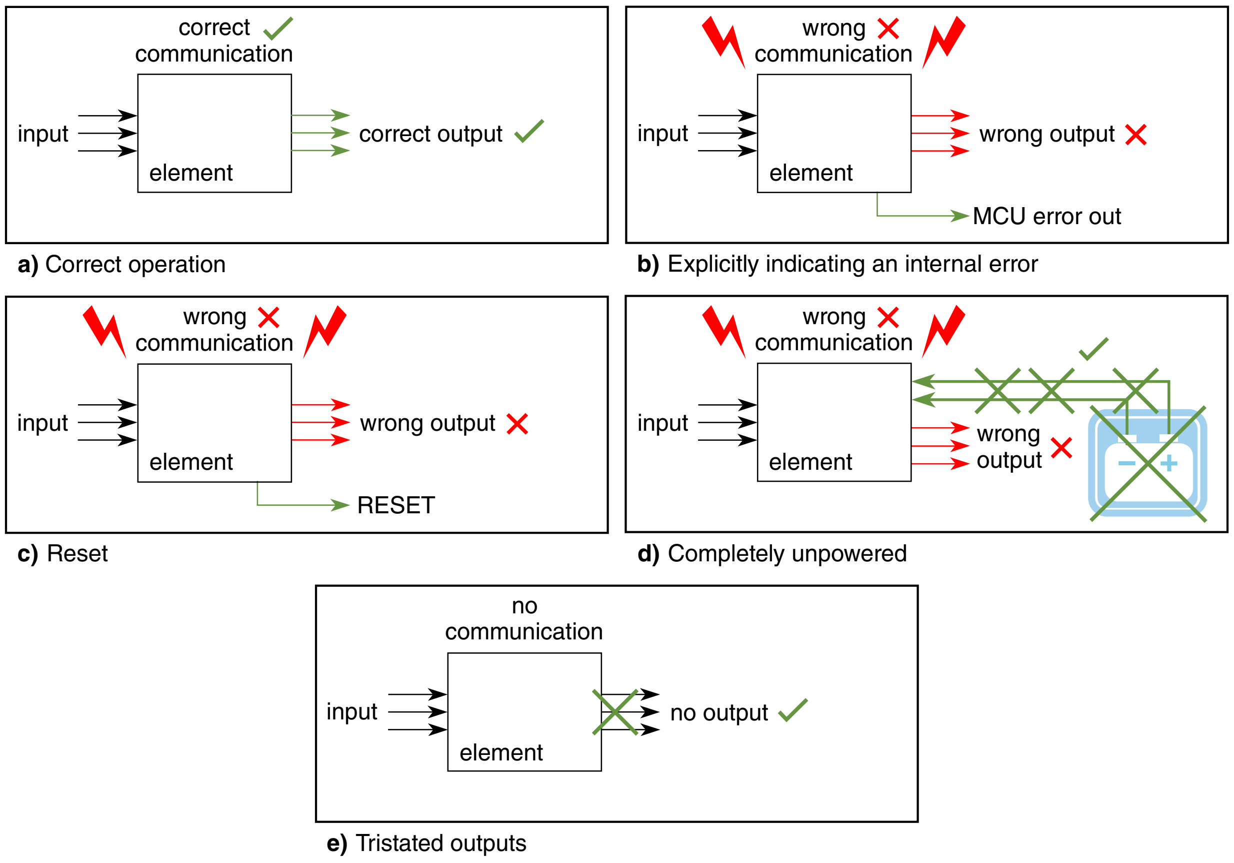

Figure 2-1. Safe state$_{MCU}$ of S32V234

2.4.2 Transitions to Safe state$_{system}$

Assumption: [SM_015] The system transitions itself to a Safe state$_{system}$ when the MCU explicitly indicates an internal error (as shown on FCCU_F0 or FCCU_F1). [end]

Implementation hint: If the S32V234 signals an internal failure via its error out signals (FCCU_F*n*), the surrounding subsystem shall no longer use the S32V234 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$_{system}$ without any additional action by the S32V234. Depending on the configuration, the system may disable, or reset, the S32V234 as a reaction to the error signal.

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

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

Assumption: [SM_018] The system transitions itself to a Safe state$_{system}$ 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]

See Consecutive resets for details.

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 S32V234.
- **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.

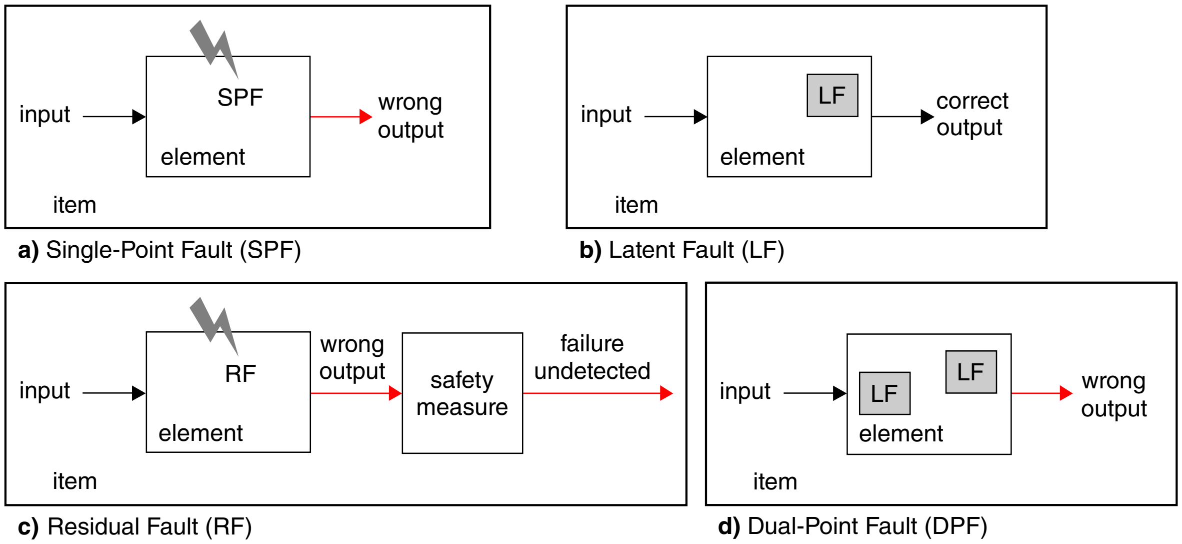

Figure 2-2. Faults

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 $\lambda_{\text{single channel}}^{N}$ ($\lambda_{\text{redundant element}} > \lambda_{\text{single channel}}^{N}$).

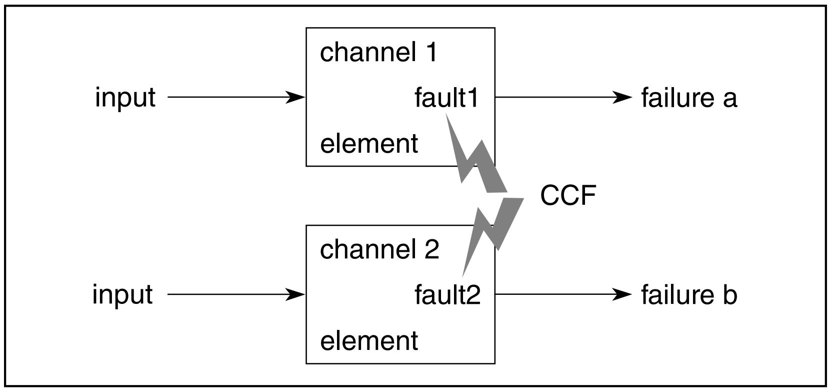

Figure 2-3. Common Cause Failures

- **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.

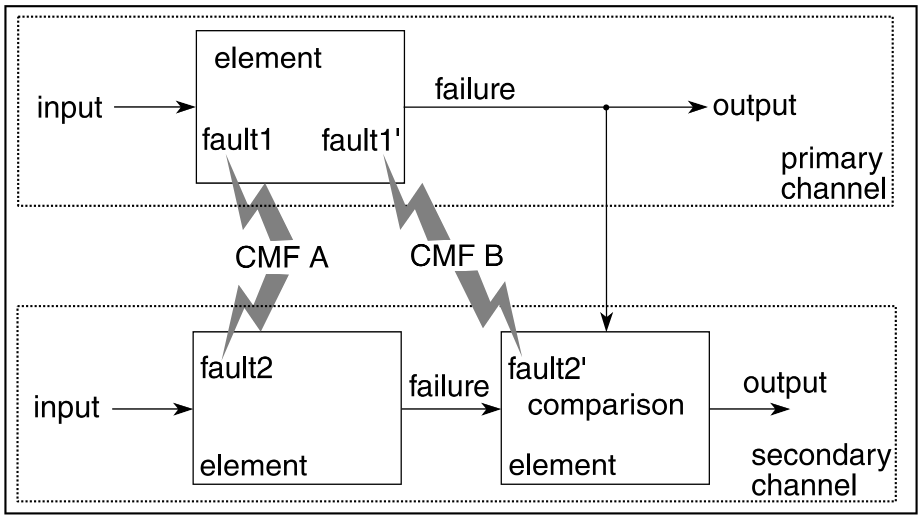

Figure 2-4. Common Mode failures

- **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).

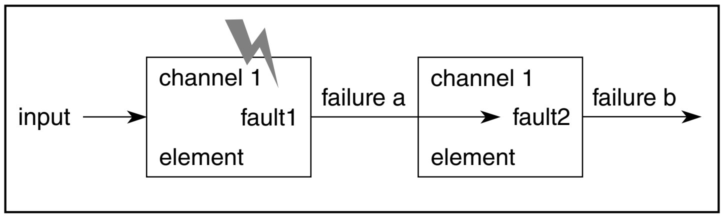

Figure 2-5. Cascading failures

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.

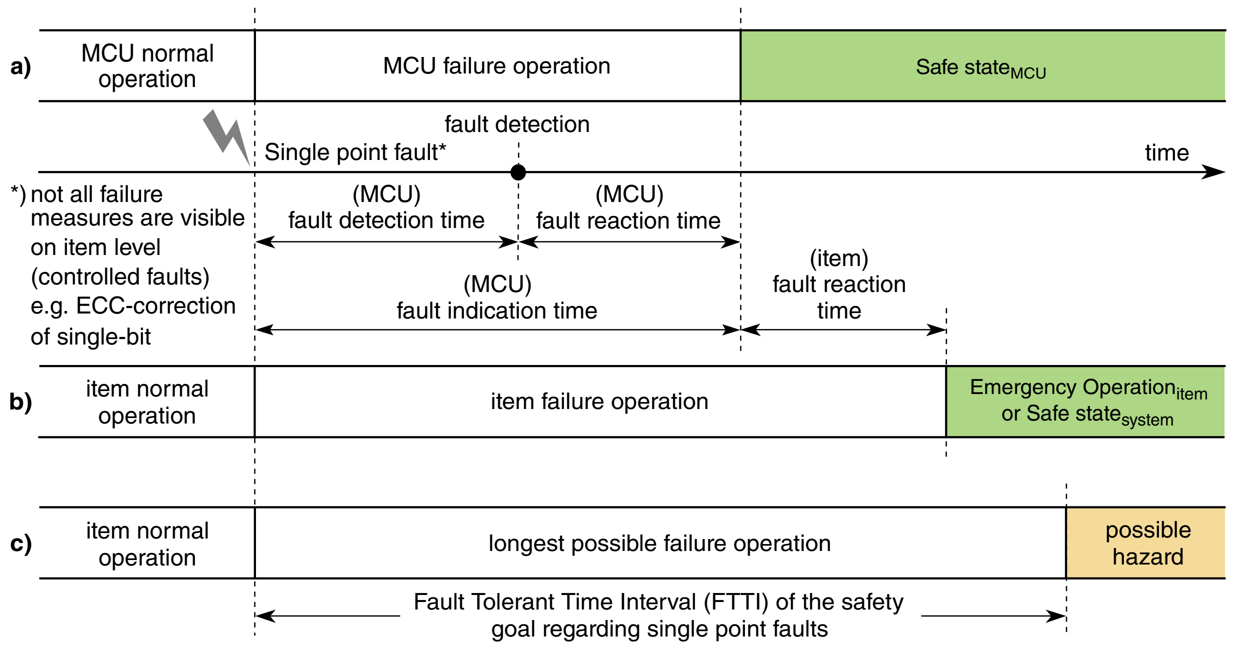
a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
fault detection
Single point fault*
time
*) not all failure measures are visible on item level (controlled faults) e.g. ECC-correction of single-bit
(MCU) fault detection time
(MCU) fault reaction time
(MCU) fault indication time
(item) 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 S32V234 is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the S32V234 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:
    - ADC recognition time is a very demanding hardware test in terms of timing. The self-test requires the ADC conversion to complete a full test. Refer to the "Self-test" section in the Analog-to-Digital Converter (SAR-ADC) chapter in the Reference Manual.

    - 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 cycles of FIRC3_CLK (nominal frequency of 16 MHz).
  - **External indication time** to notify an observer about a failure external to the S32V234. 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 (FCCU_F*n*) frequency. With an FIRC3_CLK frequency of 16 MHz, the error out frequency is 434Hz (with default parameters). Therefore, the maximum indication delay is 1.15 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, Interconnect contention, register saving, and so on).

The sum of the S32V234 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.

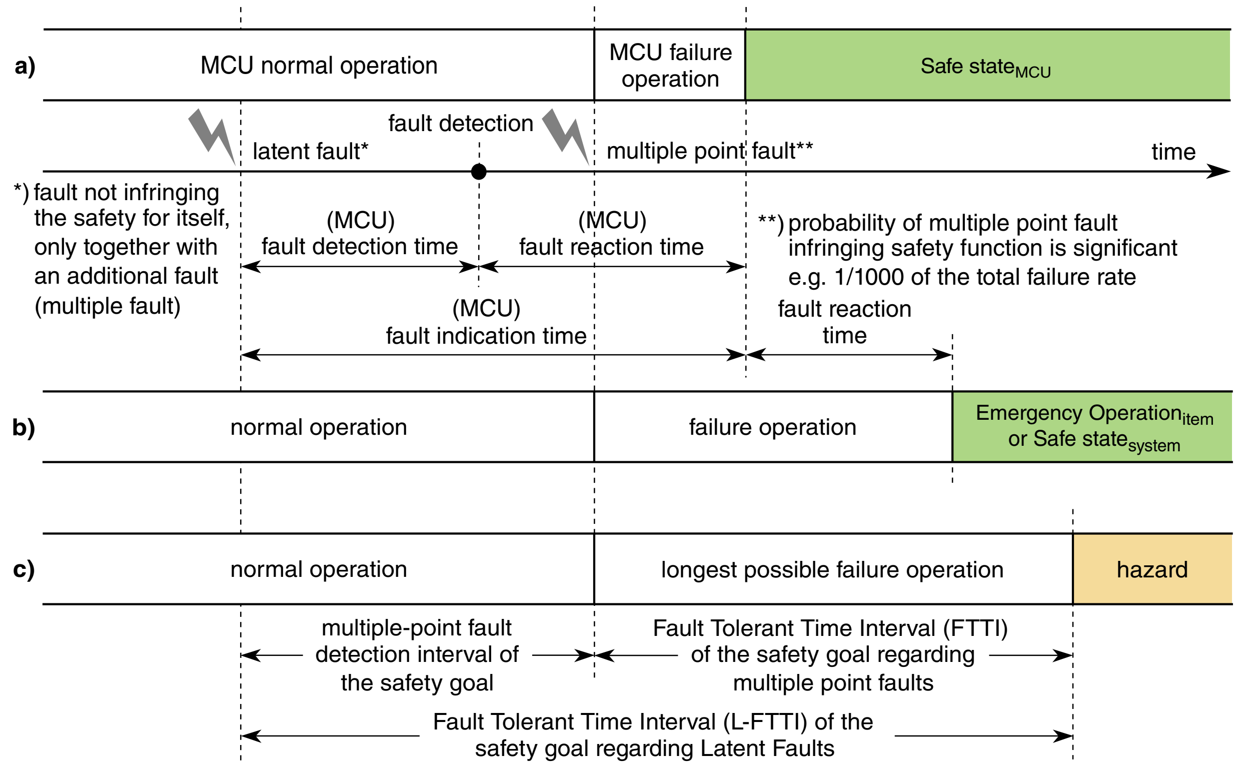
a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
latent fault*
fault detection
multiple point fault**
time
*) fault not infringing the safety for itself, only together with an additional fault (multiple fault)
(MCU) fault detection time
(MCU) fault reaction time
**) probability of multiple 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 FCCU_F*n*, forcing the outputs of the S32V234 to a high impedance state or by assertion of reset.

Assumption:[SM_212] It is assumed that the MCU will go through a complete power-up/power-down cycle within the L-FTTI. [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 S32V234 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 S32V234.

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 S32V234. The FCCU provides two error output signals (FCCU_F0 and FCCU_F1) 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 FCCU_F*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 FCCU_F*n* outputs will transition to the operational state only by software request.

At least one of the FCCU_F*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, FCCU_F0 = 0 and FCCU_F1 = 1 and vice-versa), the application software can configure that FCCU_F*n* signal that will be high to indicate the operational state (see Error Out Monitor (ERRM) for details on requirements for connecting FCCU_F*n* to external devices).

Chapter 3
MCU Safety Concept

3.1 General concept

The S32V234 has an integrated safety concept targeting safety-related systems requiring high safety integrity levels. In addition to the provided hardware mechanisms, software mechanisms may be required to detect random hardware faults. In general, safety integrity is achieved in the following ways (some software/system measures are optional):

Table 3-1. Safety Measures reducing risk of Single Point Faults

Topic	Safety Measures
Processing units (ARM Cortex CA53 Core)	Structural software based core self-test (SCST) achieving high diagnostic coverage
	(optional) Software based temporal redundant processing
	Spatial hardware redundancy (separated clusters) supporting optional software redundancy (e.g. reciprocal comparison)
Program sequence monitoring	Dedicated watchdog with time window and access code (logical program flow monitoring) for each CPU
Image processing units (APEX2)	Multiple instances of processing units (multiple data PU)
	Shared instruction decoding validated by software based self-test
	(optional) Software based temporal redundant processing (video frames)
	Spatial hardware redundancy (multiple APEX) supported by software (reciprocal comparison)
Clock generation and distribution	Diverse hardware redundancy (FIRC/FXOSC) with crystal independent safety clock
	Clock-supervision units
Supply Voltage	Voltage supervision for undervoltage (LVD) and overvoltage (HVD)
	Voltage monitoring with ADC
On-chip communication	EDC for data and address buses
Interrupt	Interrupt request hardware timeout

Table continues on the next page...

Table 3-1. Safety Measures reducing risk of Single Point Faults (continued)

Topic	Safety Measures
	Software based time-slot monitoring
DMA	eDMA and FastDMA in lock-step
	Software based time-slot monitoring
	Address/access supervision
	Data block signature (CRC...) when moving data between SRAM and DRAM
Timers	Two Instantiations of Periodic Interrupt Timer: PIT0 supports 6 channels, PIT1 supports 4 channels
Communications (Ethernet, CAN, Flexray, SPI…)	Hardware redundancy (SPI, CAN FD)
	Software based safety protocol (Ethernet, CAN FD, Flexray, PCIe and SIPI)
Volatile memories - external (DDR DRAM...)	Subrange protected by ECC (SEC/DED/TED) with single bit correction and double/triple-bit error detection per byte
	ECC includes address protection
	Block code for image frames (software measure supported by hardware CRC)
Volatile memories - integrated (SRAM, cache...)	ECC (SEC/DED) or EDC for safety relevant memories
	Address protection for selected memories
Non-volatile memories - external (Flash)	Block code (software measure supported by hardware 128-bit AES)
	Hardware support for single-bit correction SEC ECC
Analog Input (SAR ADC)	Built-in self-test integrated in SAR ADC
Diverse Safety Channel	Independent and diverse state machine (Safe State Engine) supervising safety relevant states (external or internal) and switching to safe state
Fault reporting	Fault collection and control unit (FCCU)
	FCCU itself supervised for no reaction

To improve the robustness and reduce the risk of latent (multiple point faults) this family of devices supports the following multiple point fault safety measures:

Table 3-2. Safety Measures reducing risk of Latent (multiple point) Faults

Topic	Safety Measures
Self-test (e.g. at start-up)	Logic built-in self-test (LBIST) to detect faults in random logic
	Memory built-in self-test (MBIST) to detect faults in internal SRAM
	Self-test for LVD/HVD and ADC
Fault injection	Fault injection possible into most safety mechanisms and fault reporting logic

Table 3-3. Safety Measures reducing interference and common cause failures

Topic	Safety Measures
Measure improving independence	Access protection (XRDC, MPU) reducing address domain interference
	Temperature sensor (TSENS) reducing temperature interference
	Design measures. For example spatial separation between: • Clusters • Redundant peripherals • Lock-stepped units
	Register protection (REG_PROT) for selected units
	Supervision of incorrect (Debug and Test) modes

3.2 DMA lockstep

The S32V234 duplicates the DMA processing elements and compares their operation in lockstep mode.

The main DMA is the primary execution element of the pair, where the checker DMA follows the execution of the main DMA in lockstep.

The processing elements which are replicated contain:

- FastDMA controller
- eDMA controller

Any operational deviations between the supervised signals will cause the FCCU to be notified of the discrepancy.

Each checker DMA does not have a direct connection to the Interconnect. All of the outputs of a checker DMA that targets the interconnect will end in an RCCU for verification, and all the inputs from the Interconnect will be split off from the main DMA Interconnect inputs.

Each Redundancy Control and Checker Unit (RCCU) compares a set of equivalent input signals provided by different sources and issues an alarm in case of a mismatch. For example, an RCCU compares the output of the checker DMA against the output of the main DMA and issues an alarm if they do not match. In case of a compare mismatch, the fault stays stable until it is explicitly cleared.

A delay of two clock cycles exists between the main and checker channel of eDMA to reduce their susceptibility to CCFs caused by power and clock disturbances (delayed lock-step).

3.3 ECC/EDC

Error correcting codes or error detection codes are used for individual protection of the majority of RAMs and the interconnect.

3.3.1 EDC for interconnect

Corruption on all data Connections between masters and slaves is detected via an Error detection scheme.

EDC (parity) bits are generated on writes by interconnect masters and checked when leaving the interconnect. On read, EDC information is added on the slave side and checked on the master side when leaving the interconnect. Data and Address use separate parity information.

This mechanism is available for NIC, CCI and AXBS.

3.3.2 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 memories are protected by EDC (Error Detection Code). Memories with no protection are either not safety-relevant or are safety-relevant but errors are detected by other mechanisms which are available or may be implemented by the application. The implementation of all RAMs is shown in the following table.

Table 3-4. ECC RAM implementations

Module	Memory	Memory column muxing factor	ECC/EDC	Address in ECC	Error Report
A53	L1 I-cache data	≥4	EDC		b
	L1 I-cache TAG	≥4	EDC		b
	L1 D-cache data	≥4	SECDED	yes	b
	L1 D-cache TAG	≥4	EDC		b
	L1 D-cache dirty	≥4	EDC		b
	TLB memory	≥4	EDC		b
	ETB memory	(don't care)	no		
	BTAC (stage 0/1)	(don't care)	no		

Table continues on the next page...

Table 3-4. ECC RAM implementations (continued)

Module	Memory	Memory column muxing factor	ECC/EDC	Address in ECC	Error Report
A53-cluster	L2 TAG RAM 256 KB	≥4	SECDED	no	b
	L2 DATA RAM 256 KB	≥4	SECDED	yes	b
	L2 DATA Victim	(don't care)	no		
	SCU snoop filter	≥4	SECDED	yes	b
M4	I-cache data	≥4	EDC		e
	I-cache TAG	≥4	EDC		e
	D-cache data	≥4	EDC		e
	D-cache TAG	≥4	EDC		e
	TCM	≥4	SECDED	no	e
	ETB memory	(don't care)	no		
HPSMI	System RAM	≥4	SECDED[1]	yes	d
FlexRay	data RAM	≥4	SECDED	no	f
	data LUT	≥4	SEDDED	no	f
CAN-FD	CAN memory	≥4	SECDED	no	f
JPEG	(all memories)	(don't care)	no		
Ethernet	TX - 1 Gbit	≥4	SECDED	no	f
	RX - 1 Gbit	≥4	SECDED	no	f
	RX parser	≥4	SECDED	no	f
eDMA	local memory	≥4	SECDED	yes	e
FastDMA	TX buffer, que memory	≥4	SECDED	no	f
QuadSPI	QuadSPI buffer	(don't care)	no		
CSE	CSE	≥4	SECDED	no	(none)
PCIe	(all memories)	≥4	SECDED	no	f
uSDHC	RX FIFO, TX FIFO	(don't care)	no		
Sequencer	CRAM	≥4	EDC		a
	PRAM	≥4	EDC		a
	KRAM	≥4	SECDED	no	f
APEX2	APU CMEM	≥4	EDC		a
	APU SMEM	≥4	EDC		a
H.264D	(all memories)	(don't care)	no		
H.264E	(all memories)	(don't care)	no		
GPU	(all memories)	(don't care)	no		
2D-ACE	Gamma RAM	≥4	SECDED	no	f
	Layer RAM	≥4	SECDED	no	f
	(all other memories)	(don't care)	no		

Table continues on the next page...

Table 3-4. ECC RAM implementations (continued)

Module	Memory	Memory column muxing factor	ECC/EDC	Address in ECC	Error Report
ISP (IPUS, IPUV)	LUT	≥4	SECDED	no	f
	IMEM	≥4	SECDED	no	c
	(all other memories)	(don't care)	no		
MMDC ECC	external DRAM (MMDC_0)	(depends on external DRAM)	SEC/DED/TED (for ECC protected region)[1]	yes	d
	external DRAM (MMDC_1)[2]	(depends on external DRAM)	SEC/DED/TED (for ECC protected region)	yes	d
VIU	FIFO-MEM	(don't care)	no		

1. Faults in the ECC logic itself of MMDC_0 are also detected during run-time (EDC-after-ECC).
2. ECC is not supported in cut 1.

Some memories use an ECC computed over data and address to detect data and addressing faults (e.g. no/wrong/multiple selection). The column muxing for RAMs without ECC or EDC is not given in the table, as this does not provide any significant benefit for safety for these RAMs.

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 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 S32V234 by the use of bit scrambling (column mux), such that 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 corrected by the ECC (see table Table 3-4 for the column mux factor of each memory).

3.3.3 ECC failure handling

Correctable and uncorrectable errors in volatile memories are signaled to the FCCU. Depending on the RAM, different error reporting paths are used.

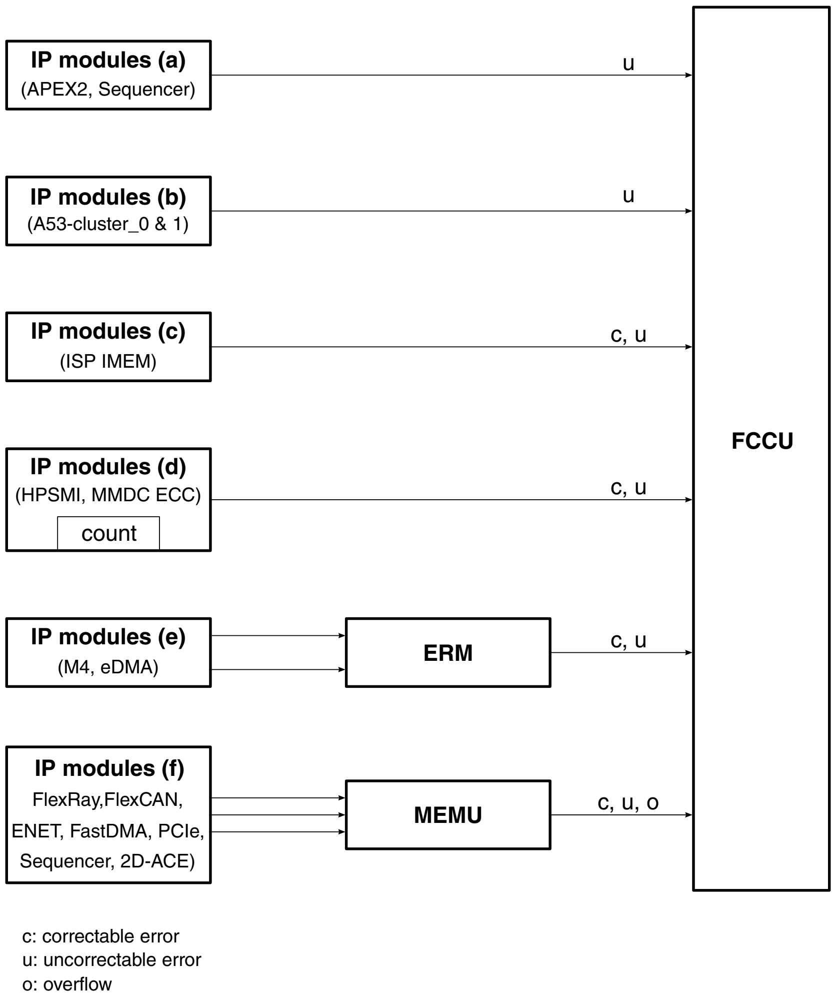

Figure 3-1. Error reporting paths

Figure 3-1 shows the different possibilities (a to f) which are available. See Table 3-4 for detailed mapping of these types to the memories.

a) Uncorrectable errors are directly reported to the FCCU (error correction not available).

b) Uncorrectable errors are directly reported to the FCCU. Correctable errors are not reported to the FCCU but the most recent error is available in a status register within the module.

c) Correctable and uncorrectable errors are signaled to the FCCU on separate failure inputs.

d) Correctable and uncorrectable errors are signaled to the FCCU on separate failure inputs. A counter exists within the module which counts error corrections.

e) Correctable and uncorrectable errors are reported to the FCCU. The most recent error is available in a status register within the ERM.

f) Correctable and uncorrectable errors are filtered by MEMU and reported to the FCCU. Errors with different error addresses are available in status registers within the MEMU.

Correctable errors should be transparently corrected without reporting to not disturb the application. This means that correctable errors for type c, d and e are typically disabled in FCCU.

3.4 Clock and power monitoring

3.4.1 Clock

The architecture consists of a redundant clock system. The functional clocks are derived from a crystal clock. In parallel, a safety clock is derived from an internal RC-oscillator which is independent of the functional clock system and is available even if the crystal clock is incorrect or is not available.

Clocks in the S32V234 are supervised by Clock Monitor Units (CMUs). The CMUs are driven by the FIRC (48 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.

3.4.2 Power

There are two types of voltage supervisors on the S32V234: Low Voltage Detect (LVD) and High Voltage Detect (HVD) monitors. Safety-relevant voltages (recommended operating voltages) are supervised for values 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 the "Power Management Controller block (PMC)" chapter in the S32V234 Reference Manual for details).

3.5 I/O peripherals

To allow a safety application to make redundant use of all I/O peripherals, they each have at least two instances, and each instance is connected to a different PBRIDGE. This means, for example, that if SPI is provided by the MCU, then two SPI modules (SPIn, SPIm) are included and connected externally through different pins. Internally, SPIn would then be connected to PBRIDGE0 and SPIm to PBRIDGE1 and they would be accessible via different addresses.

The arrangement of I/O peripherals on two PBRIDGEs, as well as further CCF prevention measures, allows redundant use of peripherals while limiting possible causes of CCFs. Redundant usage includes usage of equivalent peripherals in a replicated way as well as usage of functionally different peripherals in, for example, feedback measurement loops. Comparison of redundant operation is the responsibility of the application software, not the safety hardware mechanism.

3.6 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 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) nor are they duplicated or spread over the PBRIDGE:

- FlexRay
- CAN
- Ethernet
- LFAST/SIPI
- LINFlexD
- PCIe

Typically, software measures for the communication controllers (also called fault-tolerant communication layer) could contain e2e CRC data protection, sender identification, sequence numbering, and an acknowledgement mechanism.

3.6.1 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 and FlexRay. 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_MSCRn[SMC] for the pins that are associated with communication controllers where this feature is needed (see the "Pin muxing" table and the SIUL2 Multiplexed Signal Configuration register description in the "System Integration Unit Lite2 (SIUL2)" chapter for details, as shown in the *S32V234 Reference Manual*).

NOTE

This feature is not available for communication controllers (PCIe, LFAST/SIPI) which have dedicated pins not using the SIUL2.

The FCCU intends to drive FCCU_F0 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 FCCU_F0 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_F0 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 FCCU_F0 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 communication module (for example, FlexCAN, FlexRay) functionality and ensure those pins do not remain in an undriven state. This feature is not available for modules that are not routed through the SIUL2 (for example, PCIe, LFAST).

3.7 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 S32V234, BIST is the main means to meet the requirement on latent faults as defined by the ISO 26262 standard. Different "flavors" of BIST are implemented in the S32V234: LBIST for digital logic, MBIST for memories, and the MCU's built-in mechanisms for testing the ADC and LVD/HVD.

The following are not covered by MBIST and may be tested on the application level if necessary:

- External DRAM
- External flash memory (accessible through QSPI)

3.8 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).

3.8.1 External error indication

Failure of the MCU is signaled to one or two pins, FCCU_F0 and FCCU_F1. In addition, one GPIO input can also serve as an error input mechanism (see Fault Collection and Control Unit (FCCU) and the FCCU configuration section in the *S32V234 Reference Manual* for details on the fault output signals).

The error indication on pins FCCU_F0 and FCCU_F1 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 value, though this might differ from the logical state if a toggling protocol is used.

3.8.2 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 S32V234 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.
- 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 FCCU_F*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, FCCU_F*n*
- Disabling the transmission capabilities of communication controllers (for example , FlexRay, 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.

3.8.3 Fault inputs

The table "FCCU Non-Critical Faults Mapping" in chapter "Chip Configuration" of the *S32V234 Reference Manual* shows the source of the fault signals and the type of fault input to which these signals are connected at the FCCU.

3.8.4 FCCU supervision (FOSU)

As the FCCU is a central component in reacting to errors, it is itself supervised even though an error in it can only cause a latent failure. This supervision is provided by the FOSU (FCCU Output Supervision Unit). The FOSU receives failure indications at the same time as the FCCU. Unless the respective failure is switched off, the FOSU will observe the outputs of the FCCU (IRQ, RESET, FCCU_F[n]). If the FCCU does not react—within a predefined interval—on one of those outputs to the incoming failure indication, the FOSU assumes the FCCU has failed and causes a reset.

The FOSU does not require any configuration by software.

3.9 Common Cause Failure measures

Various measures are included to prevent CCFs from endangering the effectiveness of the intended safety mechanisms. These measures include physical separation of the components on the die, routing restrictions and supervision of clock, power, temperature, test and debug signals.

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.

3.10 Safe State Engine

The Safe State Engine can be used to perform diagnostics to generate fault signals if preconditions are not met.

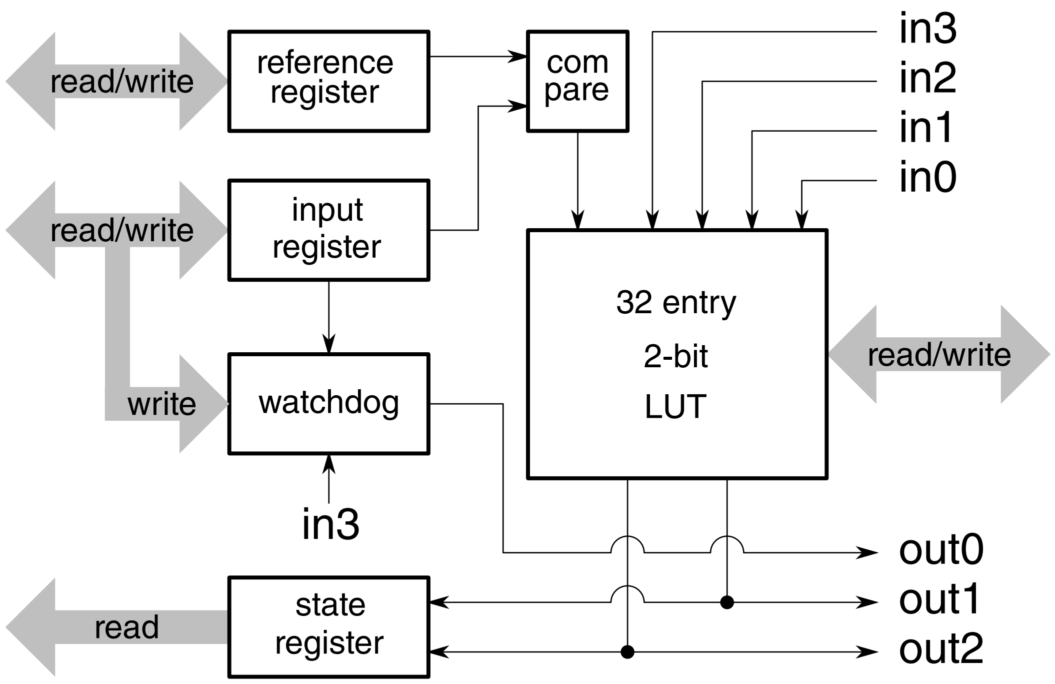

Figure 3-2. Safe State Engine overview

The Safe State Engine receives inputs from external pins or via input register SSE_IR, which may be written repetitively by the CPU or eDMA. The currently programmed value is compared with a reference (compare register SSE_CR). A watchdog provides notification if the input register is not updated in time or if the expected change of an input pin is missing. These inputs are used to generate a two-output signal from a fully programmable look-up-table (LUT). The output signals are available on external pins and are also used for fault reporting.

The Safe State Engine is also protected against unwanted configuration changes. A key advantage of this module is that, once configured, it reacts completely independently from the software.

3.11 Operational interference protection

Being a multi-master system, the S32V234 provides safety mechanisms to prevent non-safety masters from interfering with the operation of safety-relevant masters, as well as mechanisms to handle the concurrent execution of software with different (lower) ASIL. Interference freedom is achieved via a hierarchical memory protection schema including:

- MMU (A53) and MPU (M4 core)
- XRDC
- Register protection

There are two Memory Protection Unit levels included in the S32V234. The Core Memory Management Unit (MMU), present on the ARM Cortex A53, 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 XRDC 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.

A local protection within the Shared Memory Interconnect (HPSMI) exists which protects from unintentional memory overwriting caused by the following modules:

- ISP

- H264D
- H264E
- JPEGD (JPEG Decoder)

3.12 Triple-voted flops (TVF)

Several registers which are safety critical but may be insufficiently covered by a safety mechanism have been manually identified and triple-voted. This means that actually three flip-flops are used per functional flip-flop with a majority vote. A bit-flip in one of these flip-flops does not have an effect and is thus controlled.

Triple-voted flops are used for selected flip-flops within the following modules:

- MC_CGM
- PLL_DIG
- FXOSC
- FCCU
- STCU
- MTR
- MCT
- MBIST logic
- Test Controller logic
- CA53/Cluster
- SRC

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 S32V234.

The S32V234 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 S32V234, it is assumed that there will be some separate means to bring the system into Safe state$_{system}$.

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 S32V234 Data Sheet in section "Absolute maximum ratings").

The external circuit will also monitor the FCCU_F*n* signals. Through a digital interface (for example, SPI), the S32V234 repetitively triggers the watchdog of the external IC. If there is a recognized failure (for example, watchdog not being serviced, assertion of FCCU_F*n*), the reset output of the external IC will be asserted to reset the S32V234. 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).

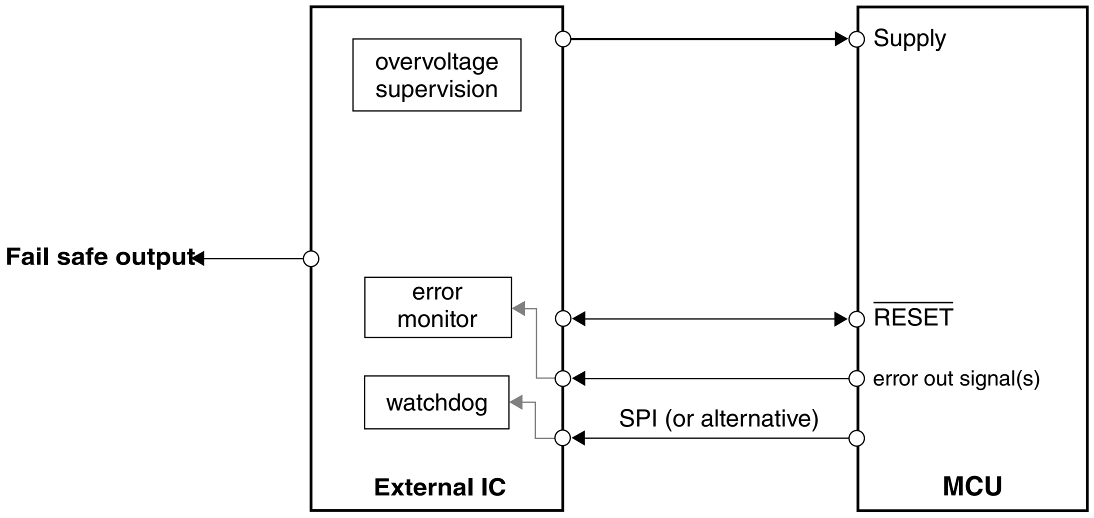

Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the S32V234 for safety-related systems.

It should be noted that failure modes of external services are only partially considered in the FMEDA of the S32V234 (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 S32V234 is considered to be in a Safe state$_{\text{MCU}}$ (for example, unpowered and outputs tristated), the system containing the S32V234 may not be compliant with the Safe state$_{\text{system}}$. A possible system level safety measure to achieve Safe state$_{\text{system}}$ 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) S32V234.[end]

Rationale: In order to bring the safety-related outputs to such a level, that a Safe state$_{\text{system}}$ 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 S32V234 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$_{\text{system}}$ within the FTTI (for example, the EXWD disconnects the S32V234 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$_{\text{system}}$ within the FTTI.[end]

The implementation of the communication between the S32V234 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, FlexRay)

4.1.1.3 Power Supply Monitor (PSM)

Supply voltages outside of the specified operational ranges may cause permanent damage to the S32V234, even if it is held in reset.

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

The *S32V234 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. [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 the Data Sheet in section "Absolute maximum ratings"). [end]

If the power supply is out of range, *S32V234* 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 *S32V234* even if kept in reset.

Implementation hint: An external and independent device may provide an over voltage monitor for the external S32V234 supplies. If the supplied voltage supply is above the recommended operating voltage range of the S32V234, the S32V234 should be maintained with no power. The external power supply monitor will switch the system to a Safe state$_{\text{system}}$ within the FTTI, and maintain it in Safe state$_{\text{system}}$ (for example, over-voltage protection with functional safety shut-off, or a switch-over to a second power supply unit).

Over-voltage on some supplies will be detected by the S32V234 itself, but system level measures might be required to maintain the Safe state$_{\text{system}}$ in case an over-voltage situation may cause damage to the S32V234.

4.1.1.4 Error Out Monitor (ERRM)

If the S32V234 signals an internal failure on its error out signals (FCCU_F0, and/or FCCU_F1), the system may no longer rely on the integrity of the other S32V234 outputs for safety functions. If an error is indicated, the system has to switch to, and remain in, Safe state$_{\text{system}}$ without relying on the S32V234. 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 FCCU_F*n* of the MCU and move the system to a Safe state$_{\text{system}}$ 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 *S32V234 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, FCCU_F0 (or FCCU_F1), is connected to a separate device.

If a fault occurs, the FCCU communicates the fault to the separate device through the FCCU_F0 (or FCCU_F1).

The functionality of FCCU_F0 (or FCCU_F1) can be checked in the following manner:

- FCCU_F0 (or FCCU_F1) read back internally.
- FCCU_F0 (or FCCU_F1) connected externally to a GPIO.
- FCCU_F0 (or FCCU_F1) uses time domain coding (for example, is active for a deterministic time interval).
- Test the ability of FCCU_F0 (or FCCU_F1) to disable system functionality (for example, measure voltage available at a motor if FCCU_F0 (or FCCU_F1) 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 FCCU_F*n* signal being used instead of using both FCCU_F*n* signals as in the previous section, is the lack of need for the separate device to compare the FCCU_F*n* signals.

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

Recommendation: If FCCU_F0, or FCCU_F1, is connected to a device not using time domain coding, verification is needed that the FCCU_F*n* signal(s) are operating correctly before execution of any safety function can start.

Rationale: To check the integrity of FCCU_F0, or FCCU_F1

To verify the functionality of a FCCU_F*n* signal, a fault may be injected into one of the FCCU_F*n* signals. The behavior of the signal can then be verified by the other FCCU_F*n* signal, or GPIO. Additionally, the fault output mode can be configured to one of the test modes to control one FCCU_F*n* as an output while the other FCCU_F*n* pin is an input or output. For example, TEST0 mode configures FCCU_F0 as an input and FCCU_F1 as an output. This test mode can be used to check the state of the FCCU_F0 input by reading FCCU_EINOUT[EIN0]. Likewise, the user can control the FCCU_F1 output by modifying FCCU_EINOUT[EOUT1].

Since the FCCU will be monitoring the system, it is sufficient to check FCCU_F0 (or FCCU_F1) within the L-FTTI (for example, at power-up) to help reduce the risk of latent faults. It is recommended that FCCU_F*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 FCCU_F0 (or FCCU_F1) 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 FCCU_F0 (or FCCU_F1) will result in the higher drive strength of FCCU_F*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 (FCCU_F0, or FCCU_F1), 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.

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 S32V234 in safety systems.

The S32V234 on-chip modules not explicitly mentioned here do not require specific safety measures to be used in safety systems. The modules that are replicated (eDMA and FastDMA) reach a very high diagnostic coverage without additional dedicated safety measures at application or system level.

5.2 Power

5.2.1 Power Management Controller (PMC)

The PMC manages the supply voltages for all modules on the device. This unit includes 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 potential failure (for correct operating voltage ranges please see the *S32V234 Data Sheet*).

To ensure functional safety, the PMC monitors various supply voltages of the S32V234 device (as seen in Table 5-1):

Assumption: [SM_144] The hardware-assisted self-test of LVD and HVD must be performed during startup. [end]

The above assumption is per the default configured in the PMC fuses.

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

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 S32V234 (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 confirm that the direct transition into reset by the MC_RGM due to an overvoltage or undervoltage event is enabled. [end]

Table 5-1. PMC monitored supplies

Supply name	Voltage rail						Safety mechanism			
	1.0	1.2/ 1.35/ 1.5	1.5	1.8	2.5	3.3	HVD	LVD	ADC [1]	Other mechanism
VDD_LV_CORE_SOC	X						yes	yes	yes	
VDD_LV_CORE_ARM	X								yes	
VDD_LV_CORE_GPU	X								yes	
VDD_LV_CSI	X									rx error
VDD_LV_PLL	X								yes	
VDD_LV_POST	X									not safety-related
PCIE_VP	X									communication error
VDD_DDR_IO		X							yes	
VDD_HV_PMC				X			yes	yes	yes	
VDD_HV_FXOSC				X				yes		
VDD_HV_LFASTPLL				X						communication error
VDD_HV_DDR				X					yes	
VDD_HV_EFUSE				X						only relevant for fusing
VDD_HV_CSI				X						rx error
VDD_HV_PLL				X					yes	
VDD_HV_ADV				X						fail of ADC self-test
VREFH_ADC				X						fail of ADC self-test
VDDIO_LFAST				X					yes	
PCIE_VPH				X						communication error
VDD_GPIO0						X		yes	yes	

Table continues on the next page...

Table 5-1. PMC monitored supplies (continued)

Supply name	Voltage rail						Safety mechanism			
	1.0	1.2/ 1.35/ 1.5	1.5	1.8	2.5	3.3	HVD	LVD	ADC [1]	Other mechanism
VDD_GPIO1				X		X			yes	
VDD_GPIO2				X		X			yes	
VDD_HV_IO_DIS				X		X			yes	
VDD_HV_IO_ETH			X	X	X	X			yes	
VDD_HV_IO_FLA				X		X			yes	
VDD_HV_IO_VIU0				X		X			yes	
VDD_HV_IO_VIU1				X		X			yes	

1. There are internal connections from the PMC to the ADC.

Over voltage of supply shall be monitored externally as described in Power Supply Monitor (PSM).

5.3 Clock

5.3.1 Phase-locked loop (5 x PLL)

The S32V234 consists of five PLLs used to generate high speed clocks. The PLLs provide a loss of lock error indication that is routed to the FCCU. Glitches which may appear on the crystal clock are filtered (low-pass filter) by the PLL.

5.3.1.1 Initial checks and configurations

After system reset, the external crystal oscillator is powered down and the PLLs are powered down. Software shall enable the oscillator. After system reset, the S32V234 uses the internal RC oscillator clock (FIRC) as its clock source (see the "Clocking" and "FIRC Digital Interface" chapters in the *S32V234 Reference Manual* and 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 (FXOSC) shall be configured as the source clock for the PLLs. [end]

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

Implementation hint: The PLLs can be configured to use the external oscillator (FXOSC) as a clock reference, or an internally provided clock reference. On the S32V234, the SRC GPR bits, GPR1[31:27], are used for this purpose.

5.3.2 Clock Monitor Unit (14 x CMU)

With the boot ROM, the intended default configuration will be that it initializes FXOSC and the PLLs before application code can run. Stuck-at faults on the external oscillator (FXOSC) are not detected by the CMUs at power-on since the monitoring units are not initialized and the S32V234 is still running on the FIRC.

The CMUs are driven by the 48 MHz internal reference clock oscillator (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 programmed range, an error signal is sent to the FCCU.

All 14 CMUs use the FIRC (48 MHz internal oscillator) as the reference clock for independent operation from the monitored clocks. Their purpose is to check for error conditions due to:

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

The 14 CMUs supervise the frequency range of various clock sources. In case of abnormal behavior, the information is forwarded to the FCCU as faults (see FCCU mapping of faults).

Assumption: [SM_080] For safety-relevant applications, the use of the CMUs is mandatory. If the modules that the CMU monitors are used by the application safety function, the user shall verify that the CMUs are not disabled and their faults are managed by the FCCU. [end]

The FCCU's default condition does not manage the CMU faults, so it must be configured accordingly.

5.3.2.1 Initial checks and configurations

Assumption: [SM_081] The following supervisor functions are required: Loss of external clock, PLL frequency higher than the (programmable) upper frequency reference and PLL frequency lower than the (programmable) lower frequency reference. [end]

Rationale: To monitor the integrity of the clock signals.

It is in general expected that SW uses the SWT so that lost clocks or significantly too slow clocks will also be detected by the SWT. See the Software Watchdog Timer section.

5.3.3 External Oscillator (FXOSC)

FlexRay and FlexCAN each feature modes in which they are directly clocked from the FXOSC.

5.3.3.1 Initial checks and configurations

Assumption: [SM_075] FlexRay and FlexCAN, both of which feature modes to be clocked directly by the FXOSC, should not make use of these modes in normal operation unless effects of clock glitches are sufficiently detected by the applied FTCOM layer. [end]

5.3.4 Internal RC Oscillator

The Internal RC Oscillator (FIRC) has a nominal frequency of 48 MHz, but frequency accuracy over the full voltage and temperature range has to be taken into account (see the *S32V234 Data Sheet*). Functional safety-related modules which use the clock generated by the FIRC are:

- FCCU
- CMU
- SWT
- SSE

In the rare case of an FIRC clock failure, these modules will stop functioning. The FCCU uses FIRC divided by three (FIRC3_CLK) which is additionally monitored by CMU_7.

5.3.4.1 Initial checks and configurations

The frequency meter of CMU_0 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.

Assumption: [SM_073] The FIRC frequency is measured and compared to the expected frequency of 48 MHz. This test is performed after power-on, but before executing any safety function. [end]

Implementation hint: 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.

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.

5.3.4.2 Runtime checks

Frequency metering of CMU_0 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 frequency metering of CMU_0 to read the FIRC frequency and compare it against the expected value of 48 MHz. In case of a mismatch, there exists the possibility of a complete failure of all safety measures. SW should then bring the system into a safe state without relying on modules driven by the FIRC (e.g. FCCU and SWT).[1] [end]

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

1. Nominal frequency of the FIRC is 48 MHz, but a post trim accuracy over voltage and temperature must be taken into account (see the *S32V234 Data Sheet*).

5.3.5 Local clocks

The following modules contain local clock-generation (local PLL):

- LFAST
- PCIe
- MIPI_CSI (for test only)

Faults within these modules are expected to be covered by the Fault-tolerant communication protocol (LFAST or PCIe) or are safe (MIPI_CSI2).

5.4 Non-volatile memory

5.4.1 External Flash

The S32V234 does not provide non-volatile internal flash memory. Flash memory can be connected via the QuadSPI or uSDHC interface.

Assumption:[SM_962] After copying the flash content into external DRAM, a CRC value shall be calculated over the array and compared with a reference.[end]

Recommendation: The CRC feature of the FastDMA is intended to perform the array integrity check at startup.

Assumption under certain preconditions:[SM_909] If code is executed out of flash during runtime, application software will implement error correction/protection for safety critical code. [end]

Assumption under certain preconditions:[SM_910] If data is stored/retrieved in flash during runtime, application software will implement error correction/protection for safety critical data. [end]

Rationale: To check that the written data is coherent with the expected data. For example, the content written to flash is read back and compared with the intended value.

5.4.2 On-The-Fly Decryption Engine (OTFAD)

The OTFAD provides hardware support for two-dimensional parity check (MDPC). The aim is to detect single-bit failures and many multiple bit failures in flash memory connected to the QuadSPI controller, with the ability to correct single-bit failures within a checked region. This feature is mainly intended to increase the availability, not primarily a measure to increase safety.

5.4.3 eFUSE

A check of the configuration programmed into eFuses can be done via the CRC functionality of the OCOTP controller during initialization of the device.

5.5 Volatile memory

All memories using no ECC are hardware-initialized by MBIST with "0". All memories using ECC without address encoding or EDC are hardware-initialized by MBIST with "0" and a valid ECC/parity value. Some memories using ECC with address encoding must be initialized by software. This is applicable for system SRAM (HPSMI, CPU memories) and the physical portion of DRAM which is protected by ECC.

Assumption:[SM_917] Application software is assumed in general not to switch into a Safe state$_{\text{MCU}}$ or Safe state$_{\text{system}}$ in case of an ECC single bit correction. [end]

5.5.1 DRAM Controller ECC (MMDC_ECC)

The DRAM controller embedds ECC data within the normal data to the DRAM (inline ECC). While this reduces the available data rate and memory space for the payload, no additional DRAM device for ECC data is required. One ECC protecatable region can be selected. Depending on the configured size of this region, all data, no data or some data is ECC protected. The ECC works on a byte basis and corrects one wrong bit within a byte of payload and detects two or three wrong bits within the data byte (SEC/DED/TED). The ECC module supports address error detection and also covers faults within DRAM-controller and phy.

Assumption:[SM_950] Volatile safety relevant DDR content shall be stored in an ECC protected memory location. [end]

5.5.2 All-X words and ECC

For some memories that are protected by ECC, a data word containing only all zeros (All-0) or all ones (All-1), including the ECC parity bits, is not legal. For these memories, a data word of All-0 or All-1 will result in an uncorrectable error. This applies to the following memories:

- PCIe
- Sequencer KRAM

- 2D-ACE RAMs
- ISP LUT
- FastDMA
- Ethernet

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. The permanent All-X failure mode is not detected by the ECC if an application reads only All-X addresses of the RAM. It is assumed that in a real application many different RAM locations (both normal and All-X addresses) are read and a dedicated test for All-X condition can be omitted.

For all other memories, All-X are legal values.

5.6 Processing modules

5.6.1 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.

5.6.1.1 JTAG Controller (JTAGC/DEBUG)

The debugging facilities of the S32V234 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 S32V234 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. The activation of debug mode, if JCOMP is low (for example, due to hardware failures), is supervised by the FCCU, and it will signal a fault condition when debug mode is entered.

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, eDMA, ENET, SIPI, 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.

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.

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

When ENET_ECR[DBGEN] = 00, the Ethernet MAC continues normal operation while the device is in debug mode.

SIPI_MCR[FRZ] controls the SIPI behavior during debug mode. If the SIPI_MCR[FRZ] = 0 (cleared), the SIPI continues serial transfers during debug mode.

Implementation hint: The accesses of the external debugger to the ETM trace unit register is assumed to be locked out by setting the OS Lock key (OSLK = 1) within the OS Lock Access Register (TRCOSLAR) during an active safety function processed.

The accesses of the external debugger to the ETM trace unit register is assumed to be double locked out by setting the Double Lock (DLK= 1) within the External Debug Power/Reset Control Register (EDPRCR) during an active safety function processed.

The application software accesses to the ETM trace unit register is assumed to be software locked by setting the Software Lock Key Value within the Software Lock Access Register (TRCLAR) during an active safety function processed.

5.6.1.2 Test mode

Several mechanisms of the S32V234 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. [end]

Implementation hint: The 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 TEST pin is not connected to V_{DD} during boot to avoid entering test mode.

The activation of test mode is supervised by the FCCU and will signal a fault condition when test mode is entered. FIRC clock-related testmode activation is intended to be covered by the frequency meter function of CMU_0, as described in section Runtime checks.

5.6.2 S32V234 configuration

Assumption:[SM_140] It is required that application software verifies that the initialization of the S32V234 is correct before activating the safety-relevant functionality. [end]

After startup, the application software must ensure the conditions described in this section are satisfied before safety-relevant functions are enabled.

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 S32V234's safety function or disable the safety mechanism due to their change.

5.6.3 System Bus Interconnect

The multi-port Interconnect switch allows concurrent transactions from any master (for example, SIPI, core, eDMA, FlexRay, and so on) to any slave (for example, memories, peripheral bridge, and so on). The Interconnect 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 Interconnect which are Not Safety Related shall have a lower arbitration priority on the Interconnect compared to Safety Related masters.[end]

5.6.3.1 Runtime checks

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

The detection of failures of the Interconnect 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 Interconnect masters. The need for periodic configuration readback depends on how stringent the control flow monitoring is implemented.

5.6.4 Interrupt Controller (GIC, NVIC)

The Interrupt Controller provides the ability to prioritize, block, and direct Interrupt Requests (IRQs). The Interrupt Controller can fail by dropping or delaying IRQs, directing them to the wrong core or handler, or by creating spurious ones. 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.

5.6.4.1 Periodic low latency IRQs

The Interrupt Monitor (INTM) can be configured to start when the interrupt request is generated and the application software can read the timer value to determine when the ISR is entered. This method can be used to determine whether the measured interrupt latency exceeds the requirements.

Assumption: [SM_099] Periodic low latency IRQs will use a running timer/counter to ensure their call period is expected.[end]

5.6.4.2 Non-Periodic low latency IRQs

Non-periodic, low latency IRQs can be handled in the methods described below.

Recommendation: Use the four high priority registers INTC_HIPRI*n*C0 to configure which interrupts to monitor and check. Program the INTC_LAT*n*C0 registers with the maximum GIC clock cycles for the monitored interrupt.

A supervisor module configured to react to any one of the IRQ signals checks that the GIC reacts with an immediate activation of the core's IRQ and the correct IRQ vector. This will only be able to supervise the highest priority IRQ.

5.6.4.3 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.

5.6.5 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.

As eDMA is a replicated module, no software action is needed to detect faults inside this module. Nevertheless, failures outside of the eDMA can lead to the eDMA behaving faulty. Such failures have to be detected by software.

5.6.5.1 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 Interconnect, as well as prevent it from copying data at a wrong point in time

Implementation hint: Possible software implementation to protect against spurious or missing interrupts, or transfer requests that over burden the MCU are as follows:

- Software counts the number of eDMA transfers triggered inside a control period and compare this value to the expected value.

5.6.5.1.1 Peripheral lake eDMA transfers

The eDMA module is replicated but the DMA Channel Mux, which maps the handshake signals of different peripherals to the eDMA, is not replicated. Each half of the DMA Channel Mux is responsible for the peripherals in its peripheral lake. Selecting peripherals which are located on two different PBRIDGEs ensures the redundancy of the channel muxes/eDMA.

Assumption:[SM_103] Software using the eDMA to transfer data between peripheral and RAM will either use eDMA to, or from, peripherals in both peripheral lakes or use other detection mechanisms to detect failures of the peripheral.[end]

For example, if DMA Channel Mux 1 is faulty and thus disturbs access to a peripheral in its lake, then an access triggered by DMA Channel Mux 0 to a peripheral in its own lake will not be faulty and thus show a deviation from the faulty transfer. PIT can be used redundantly (PIT0 for channel mux 0 and PIT1 for channel mux1).

5.6.5.1.2 Non-replicated eDMA transfers

In cases where the eDMA is used to transferred 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 DMA 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 DMA 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]

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 single PIT from triggering an unexpected eDMA transfer

5.6.6 Redundancy Control Checking Unit

The task of the Redundancy Control Checking Unit (RCCU) unit is to perform a cycle-by-cycle comparison of the outputs between the master and checker FastDMA units and the master and checker eDMA units, respectively. The error information is forwarded to the FCCU. The RCCUs are always enabled.

5.6.6.1 Initial checks and configurations

The use of the RCCU is indispensable, and is automatically managed by the S32V234. The RCCU cannot be disabled by application software during runtime. Consequently, the respective FCCU input should not be disabled.

5.6.7 Reset Generation Module (MC_RGM)

5.6.7.1 Initial checks and configurations

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

Recommendation: To enable critical events to trigger a reset sequence, the MC_RGM's Functional Event Reset Disable register should be written with zeros (MC_RGM_FERD = 0). If the customer wants to exclude particular critical events from triggering a reset sequence the corresponding bit in the MC_RGM_FERD register should be set (= 1). On S32V234, MC_RGM_FERD is only available for JTAG functional reset.

At any point, customer software can initiate a functional reset sequence or a destructive reset sequence. 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 *S32V234 Reference Manual* for details).

5.6.7.1.1 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 S32V234 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. A similar mechanism to detect a loop of continuous destructive resets is implemented in the MC_RGM. When the destructive reset counter reaches the programmed value, the MCU will be held in reset until the next power-on reset. The destructive reset counter can be cleared by software or by a power-on reset.

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

Assumption: [SM_930] Safety Software shall reset the destructive reset counter everytime after a reset when it is convinced 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]

5.6.8 Extended Resource Domain Controller (XRDC)

As a multimaster, concurrent bus system, the S32V234 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 on the A53 with the appropriate ASIL. This software shall use the XRDC to block write access to the XRDC configuration registers for all other masters. [end]

5.6.9 Watchpoints

Assumption: [SM_905] Watchpoints (DRAM and SRAM) shall not be used during safety application. [end]

Watchpoints are supervised by the FCCU to detect any activation during normal operation.

5.6.10 Cortex A53 core cluster

Dependent on the usage and ASIL of the safety goal(s), different techniques are appropriate for detection of faults on the core cluster.

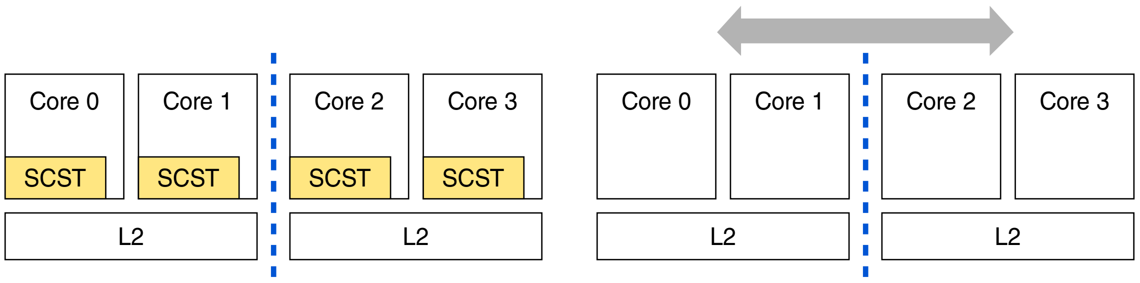

Figure 5-1. A53 core cluster: a) self-test b) redundant code execution

5.6.10.1 Core self-test

During code execution, core self-test software (for example, Structural Core Self Test - SCST) should be executed to detect failures in the A53 cores. Within the FTTI, the processing unit periodically runs code that tests the functionality of the processing unit. The result of this test was calculated offline during development and compared with the real-time value.

NXP develops core self-test software which executes at runtime on each CPU with high diagnostic coverage and low performance impact. Please contact NXP for details.

Assumption under certain preconditions: [SM_959] A software-based self-test is periodically executed within the FTTI on each A53 core. [end]

5.6.10.2 Redundant code execution

Code may be executed redundantly on two cores to detect faults during code executions. Different approaches, ranging from fully replicated processing to simple cross-check, may be possible. Redundant execution, evaluation and error handling are handled by application. To minimize the potential of dependent failures, always two cores from different cluster must be used.

Assumption under certain preconditions: [SM_961] If code is executed redundantly, the two cores are not within the same cluster. [end]

5.6.10.3 Freedom from interference

Both clusters are physically separated and interference is low due to the structural separation with layout restrictions, two separated L2 caches, and the possibility to exclude cache coherency.

Assumption under certain preconditions: [SM_963] The caches of the two clusters are not coherent for critical data which is used for redundant calculations. [end]

Assumption: [SM_092] The operating system (or other SW) on each core shall use the MMU to achieve freedom from interference between different software parts. [end]

5.6.11 Cortex M4

Depending on the usage and ASIL of the safety goal(s), different techniques are appropriate for the detection of faults if the M4 is used in a safety-relevant manner.

5.6.11.1 Core self-test

During code execution, core self-test software (for example, Structural Core Self Test - SCST) should be executed to detect failures in the M4 core. Within the FTTI, the processing unit periodically runs code that tests the functionality of the processing unit. The result of this test was calculated offline during development and compared with the real-time value.

NXP develops core self-test software which executes at runtime on each CPU with high diagnostic coverage and low performance impact. Please contact NXP for details.

Assumption under certain preconditions:[SM_959] A software-based self-test is periodically executed within the FTTI on the M4 core. [end]

5.6.11.2 Black-channel

Assumption on certain preconditions:[SM_931] If the M4 core is used for safety-critical communication (for example, CAN), the M4 core can be used as a black channel and the message can be encapsulated into a fault-tolerant communication protocol outside of the M4. Precautions against failures will be made on the system level with a fault-tolerant protocol (as described for the communication controller). [end]

5.6.12 System timer module (STM)

5.6.12.1 Runtime checks

In case a failure in the System Timer Module (STM) causes a violation of the safety goal, one of the two conditions below shall be satisfied when the STM is 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.

5.6.13 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 mapping of faults).

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. [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.

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 S32V234 provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudo-random key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

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

Two service procedures are available:

- A fix service sequence represented by a write of two fix 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.

5.6.13.1 Runtime checks

Recommendation: 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.

5.6.14 Periodic Interrupt Timer (PIT)

5.6.14.1 Runtime checks

Assumption: [SM_107] When using PIT module, the PIT module 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

5.6.15 Cyclic Redundancy Checker Unit (CRC)

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 S32V234.

5.6.15.1 Runtime checks

Parts of the S32V234 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 safety-relevant configuration registers shall be checked at least once per FTTI to verify their proper content if a wrong setting is not detected by other means. [end]

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: To verify the content of the S32V234 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.

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.

5.6.16 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 the on-chip safety mechanisms
- Configurable and graded fault control:
    - Internal reactions
        - No reset reaction
        - IRQ
        - NMI (Non-maskable Interrupt)
        - Functional Reset
    - External reaction (external failure reporting using FCCU_F*n*)

The FCCU is checked by the FCCU Output Supervision Unit (FOSU) which provides a secondary path for failure indication and reports to the Reset Generation Module (MC_RGM). The FOSU only causes a reset when the FCCU does not react to the incoming failure indication. The FOSU cannot be configured in any way, but it defines a maximum time (8000 FIRC3_CLK cycles) that the FCCU can be held in the configuration state.

The table "FCCU Non-Critical Faults Mapping" in chapter "FCCU" of the *S32V234 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, FCCU_F[0] and FCCU_F[1], through which critical failures are reported. When the device is in reset or unpowered, these outputs are tristated.

FCCU_F[*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$_{\text{system}}$ condition within the FTTI (for example, the separate device disconnects the S32V234 device or an actuator from the power supply).

5.6.16.1 Initial checks and configurations

Besides the possible initial configuration, no intervention from the S32V234 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 is configured. [end]

Rationale: Maintain the device in the Safe state$_{\text{system}}$ 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 the S32V234 signals an internal failure via its error out signals (FCCU_F[*n*]), the system can no longer safely use the S32V234 safety function outputs. If an error is indicated, the system has to be able to remain in Safe state$_{\text{system}}$ without any additional action from the S32V234. Depending on its functionality, the system might disable or reset the S32V234 as a reaction to the indicated error.

5.6.16.2 Runtime checks

If the S32V234 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$_{\text{system}}$ criteria.

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$_{\text{MCU}}$. 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$_{\text{system}}$.

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 means 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.

5.6.16.3 Error reporting path tests

It is possible to use fake fault injection to check the correct operation of several reporting paths. The FCCU fault input table in the reference manual shows more details about injecting errors.

It should be noted that LBIST covers the logic of the error reporting path as long as it does not cross an LBIST partition boundary. If that happens, a small amount of logic remains uncovered by the LBISTs, which is also tested by error reporting path tests.

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 memory to check the reporting of such errors through the MEMU to the FCCU. Software can program two patterns into RAM using the ECC bypass mechanism. If one single-bit and one double-bit error pattern is programmed, this can be used to test the error reaction path either at startup or during operation. It is sufficient that the two memory locations be programmable once per trip time (or once after the RAM content has been reset).

5.6.17 Memory Error Management Unit (MEMU)

The MEMU collects and reports error events associated with ECC logic used on some internal SRAMs. The MEMU stores the addresses where ECC errors occurred. The MEMU also reports whether the error is correctable vs. uncorrectable. New correctable errors, and each uncorrectable error (even if known), will cause a report to the FCCU.

All errors the MEMU collects are stored in reporting tables that are accessible through the MEMU register interface.

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

5.6.18 Built-in Hardware Self-Tests (BIST)

Built-in hardware self-test (BIST) 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 destructive, it shall be executed ahead or after a reset (destructive reset or external reset).

The overall control of the LBISTs and MBISTs is provided by the Self-Test Control Unit (STCU2).

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

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

- STCU2_LBSSW to determine which online LBISTs failed
- STCU2_LBESW to determine which online LBISTs did not finish
- STCU2_MBSSWn to determine which online MBISTs failed
- STCU2_MBESWn to determine which online MBISTs did not finish
- STCU2_ERR_STAT to check for internal STCU2 failure

If there is an LBIST failure, or an uncorrectable memory reported by MBIST, the application must not start (however, a degraded mode upon user discretion may be possible). If a MBIST detects an ECC correctable error, SW should continue execution. To distinguish correctable and uncorrectable permanent errors in memory, diagnostic mode of MBIST is available. NXP develops MBIST manager software to support this determination.

5.6.18.1 Scheduling

The Logic BIST (LBIST) and SRAM BIST (MBIST) runs during initialization (during boot) and can be run during shutdown, if configured appropriately and triggered by software.

NOTE

In principle MBIST can be run at any time, but will not restore the altered memory content.

NOTE

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

The ADC BISTs run during initialization (during boot) and optionally during normal operation, but software actions are required to run those tests (see Analog to Digital Converter (ADC)).

The LVD/HVD BISTs run during initialization (during boot), but software actions are required (see Power Management Controller (PMC)).

Assumption under certain conditions: [SM_914] If self test is executed during shutdown, it is the responsibility of the application software to handle and store self test results. [end]

For example, online self test is triggered and it resets the MCU on completion. The core wakes up, checks the result, and writes it into external flash.

5.6.19 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 S32V234 is protected from errant register changes made by other User mode processes.

In addition, some selected modules are 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] Configuration registers, and registers that are not modified during application execution, should be protected from unintended software write accesses (e.g. with a Hard Lock Protection) if the module supports register protection. [end]

Rationale: To reduce the risk of accidental writes to configuration registers affecting the execution of the S32V234's safety function or disabling the safety mechanism due to their change.

5.6.20 Temperature Sensor (TSENS)

The S32V234 has one temperature sensor. The temperature sensor generates a digital number which is proportional to the absolute current junction temperature of the device and a digital output that signals when the temperature sensor detects a junction temperature above a selectable temperature threshold.

Temperatures that are outside of the allowable range are handled as follows:

- FCCU failure generation according to the defined high temperature point

Recommendation: The potential for over-temperature operating conditions need to be reduced by appropriate system level measures. Possible measures could include:

- Inhibiting functional safety using a thermal fuse.
- Several levels of over-temperature sensing and alarm triggering.
- Connection of forced air cooling and status indication.

5.6.21 Wake-Up Unit (WKPU) / External NMI

Assumption under certain conditions:[SM_126] If external NMI is used as a safety mechanism, it is required to implement corresponding system level measures to detect latent faults in the WKPU. [end]

Rationale: To test the WKPU for external NMIs.

Implementation hint: To test 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.

5.7 Processing units for vision

Many processing steps in vision processing modules are robust to withstand transient fault effects. Those modules are listed in the "Core Vision" group in the attached "S32V234 Module Classification" spreadsheet. For such cases, transient faults that occur once and subsequently disappear are considered safe.

Faults where a vision module is not operable at all (e.g. black-picture, frozen-picture) or where it processes data permanently in a wrong way must be detected. Such behavior may be caused by a permanent fault or a transient fault with permanent effects. Functional tests are intended to detect such cases. A known test-pattern is processed when a module is unused (e.g. in the vertical blanking interval) and the result is compared with an expected value.

Some faults in data which is eventually shown on a screen (e.g. surround-view application) are either not seen by the driver or the picture may be massively corrupted so that the driver does not rely on it (perceived fault). For frozen images, please refer to the 2D-ACE/DEC200 section.

Assumption: [SM_957] Inherent safety of vision data is assumed for transient faults within vision data for ISP, APEX2, GPU, H264D, H264E, JPEGD, MIPI-CSI2, and VIU. If this assumption is not applicable in a particular application, additional mechanisms may be required. [end]

5.7.1 Video Decoder H.264

The fault detection for the H.264 decoder is the responsibility of the application software. For many applications, checking the transmission path for permanent errors and correct picture transmission once within the FTTI is sufficient, while transient errors (which only persist for one or a few frames) may not lead to a dangerous situation.

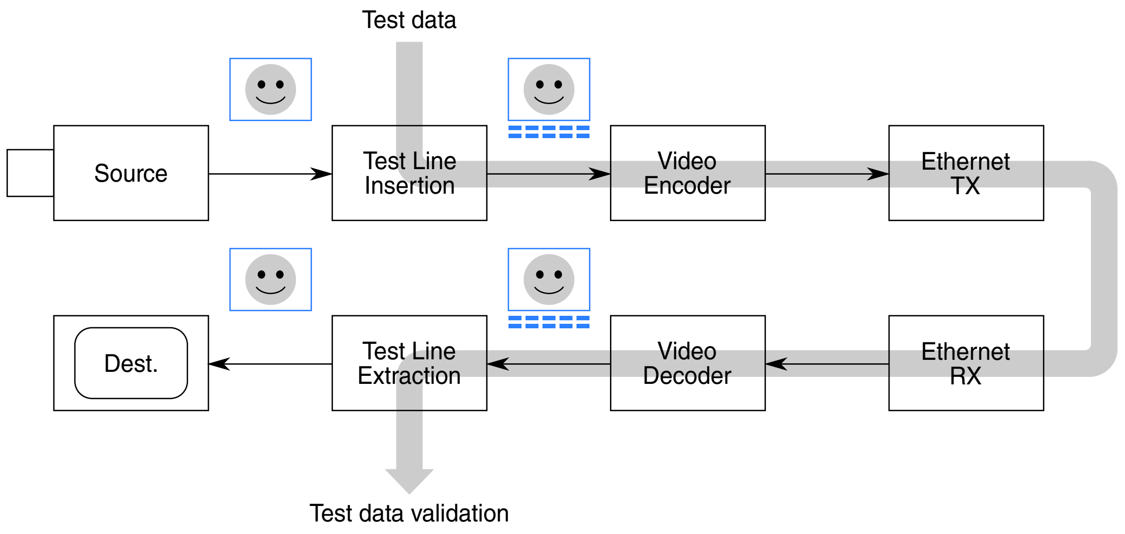

Figure 5-2. Encoder/decoder

Assumption:[SM_938] Application SW will encode a suitable pattern into the H.264 encoded data stream. [end]

Implementation hint: If the encoded stream comes from another S32V234, refer to 'Testline' section of the H264 Encoder (H264_ENC) chapter in the Reference Manual for provisions to support encoding the predefined information.

Assumption:[SM_939] Predefined information shall be decoded from the video data stream. [end]

Implementation hint: Refer to 'Testline' section of the H264 Decoder (H264_DEC) chapter in the Reference Manual for provisions to support extracting the predefined information.

Assumption:[SM_940] Application SW will check the decoded information. In case of a mismatch, H.264 picture path should not be considered correct and appropriate system level action must be taken. [end]

From a functional safety point of view, the high-intra mode is preferred compared to the contrained baseline mode of the H.264 decoder. By nature of the encoded signal, the decoding latency in high-intra mode is much lower, which allows shorter FTTI. In addition, the risk for frozen images and the error propagation among several pictures is much lower.

Assumption:[SM_941] When using constrained baseline mode, at least one I-frame shall be transmitted from the encoder within the FTTI. [end]

5.7.2 Video Encoder H.264

The fault detection for the H.264 encoder is the responsibility of the application software. For many applications, checking the transmission path for permanent errors and correct picture transmission once within the FTTI is sufficient. Refer to the Video Decoder H. 264 section.

Assumption:[SM_942] Predefined information shall be encoded in the video data stream. [end]

Implementation hint: Refer to the 'Testline' section of the H264 Encoder (H264_ENC) chapter in the Reference Manual for provisions to support encoding the predefined information.

Assumption:[SM_944] Application SW will check the decoded information. In case of a mismatch, the H.264 picture path should not be considered correct and appropriate system level action must be taken. [end]

5.7.3 Video Decoder JPEG

The fault detection for the JPEG decoder is the responsibility of the application software. For many applications, checking the transmission path for permanent errors and correct picture transmission once within the FTTI is sufficient.

Assumption:[SM_945] Application SW will encode a suitable pattern into the JPEG encoded data stream. [end]

Assumption:[SM_946] Application SW will check the decoded information. In case of a mismatch, the JPEG picture path should not be considered correct and appropriate system level action must be taken. [end]

Implementation hint: Refer to the 'Testline Feature' section of the JPEG Decoder (JPEGD) chapter in the Reference Manual for provisions to support extracting the predefined information.

5.7.4 VIULite

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.

Assumption:[SM_947] Software will periodically test for a valid input signal. [end]

Assumption:[SM_948] Software will check for FIFO under/overflows. [end]

5.7.5 MIPI-CSI2

Assumption:[SM_925] Software will periodically (once per FTTI) test for a valid input signal, using the status information provided by the MIPI-CSI2 module. [end]

Assumption:[SM_928] The image stream will contain changing embedded data generated by the source. The changing embedded data will be checked for plausibility to detect failures on the path from camera to MIPI-CSI2 output. [end]

5.7.6 Sequencer

A messaging channel between the M0+ core of the sequencer and the A53 cluster exists via PRAM.

Assumption:[SM_924] A timeout supervision and simple flow monitoring will be used on the A53 core to detect a stalled sequencer cpu. [end]

5.7.7 ISP

Assumption:[SM_919] The IPUs will be tested for permanent faults by SW. [end]

Assumption:[SM_920] Known picture content will be processed and compared to a known reference once within FTTI. [end]

Preferably, FastDMA should be used to calculate a CRC of the generated result.

Assumption:[SM_922] The comparison shall be done with a HW element independent from the ISP/Sequencer (e.g. A53). [end]

Implementation hint: One or more additional test-line(s) per frame may be added (e.g. at the end of a picture) which are processed by the IPUs.

5.7.8 GPU

Assumption:[SM_955] The GPU will be tested for permanent faults by SW. For example, known picture content will be processed and compared to a known reference. [end]

Assumption:[SM_956] The evaluation of the result will be done with a HW element independent from GPU (e.g. A53). [end]

Assumption:[SM_958] A timeout supervision will be used to detect a stalled GPU engine. [end]

Assumption:[SM_965] It is assumed that failures other than frozen images will be perceived by the driver. [end]

5.7.9 APEX2

Assumption:[SM_951] The APEX2 will be tested for permanent faults by SW. For example, known picture content will be processed and compared to a known reference. [end]

Assumption:[SM_952] The evaluation of the result will be done with a HW element independent from APEX2 (e.g. A53). [end]

Assumption:[SM_954] A timeout supervision will be used to detect a stalled APEX engine. [end]

5.7.10 2D-ACE/DEC200

When pictures are stored in memory, various system failures within the image path may cause a sequence of images to stop, resulting in a frozen image condition. Such failures may comprise camera lock-ups, communication faults, storage failures, etc. If used in a rear mirror application, the driver is no longer able to look back, potentially without noticing this. Thus it is important to inform the driver if the video display fails.

Assumption:[SM_933] A frozen image detection shall be implemented. [end]

Implementation hint: Software can store the checksums of the last n frames. n should match at least the minimum number of potentially stored image frames in the data path. A counter representing the percent of images that are shown multiple times (i.e. not updated) can be implemented in software. This is a somewhat fuzzy approach which prevents the system from triggering an error in the rare case the incoming image is really identical to the previous one (or there is, by pure chance, the same identical checksum for different pictures). Based on the counter value and an additional threshold, the software shall decide whether the system is actually in an error state, and if so then the driver shall be informed, for example, by a warning message or by switching off the display.

Failures of the frame-buffer compression (DEC200) will be detected by visual distortion and will not lead to a frozen image.

Assumption:[SM_936] It is assumed that failures other than frozen images will be perceived by the driver. [end]

The HW cursor layer of the 2D-ACE should not be used for applications which display safety relevant information on the screen.

5.8 Peripheral

Assumption:[SM_911] The integrity of PERIPHERAL elements will be mainly assured by application-level mechanisms (for example connecting one sensor to different Input/ Output modules, sensor validation by sensor fusion) which is supported on the hardware level by a replication of potentially safety-relevant Inputs/Outputs with mechanisms reducing the risk of common mode faults. The bus bridges from high speed system bus to low speed peripheral bus therefore is replicated. [end]

5.8.1 Communications

5.8.1.1 Redundant communication

Parts of the integrated SPI and LINFlex communication controller do not on their own provide the functional safety integrity IEC 61508 series 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.

Recommendation: 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
- I2C

SPI and LINFlexD do not have special functional safety mechanisms other than what is included into them by their protocol specifications. The system level communication architecture needs to provide the functional safety mechanisms on the interface of the modules to meet functional safety requirements.

5.8.1.2 Fault-tolerant communication protocol

Portions of the integrated FlexRay, 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.

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, may be required for the following:

- FlexRay Communication Controller
- FlexCAN Communication Controller
- Universal Asynchronous Communication Controller (LINFlexD)
- PCIe
- Ethernet
- LFAST/SIPI

Communication interfaces do not have specific functional safety mechanisms for single-point faults other than what is included in their protocol specifications or ECC protection of SRAM where applicable. 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').

Assumption: [SM_051] It is assumed that communication over high-bandwidth interfaces is protected by a fault-tolerant communication protocol. [end]

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.

5.8.1.3 Ethernet assumptions

Assumption:[SM_913] It is assumed that the delay of the Ethernet transmission does not lead to an unacceptable delay of picture information and will be included in FTTI calculation. [end]

5.8.2 System Integration Unit Lite2 (SIUL2)

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 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 replicated peripherals (for example, FlexTimer) to implement multiple independent and different channels.

5.8.2.1 Digital I/O

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

5.8.2.1.1 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).

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.

5.8.3 Analog to Digital Converter (ADC)

Parts of the Analog-to-Digital Converter (ADC) of the S32V234 do not provide the functional safety integrity on independently to achieve high functional safety integrity targets. Therefore, system level measures are required.

5.8.3.1 Initial checks and configurations

Assumption under certain conditions: [SM_130] When Analog-to-Digital Converter (ADC) of the S32V234 are used in a safety function, suitable system level functional safety integrity measures must be implemented once per L-FTTI. [end]

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

Implementation hint: After reset (external reset or destructive reset), but before executing any safety function, the following hardware BISTs may be executed by the application software to detect latent faults:

- SUPPLY SELF-TEST – (algorithm S) includes the conversion of the internal bandgap, 3.3V analog supply, and the ADC VREF voltages
- CAPACITIVE SELF-TEST – (algorithm C) includes a sequence of test conversions by setting the capacitive matrix

5.8.4 Glitch filter

An analog glitch filter is implemented on the reset signal of the S32V234. A selectable (WKPU_NCR[NFE0]/WKPU_NCR[NFE1]) analog glitch filter is implemented on the NMI input. External interrupt sources can be configured to be used with any chip GPIO. Interrupt sources (1 to 32) can be configured to have a digital filter to reject short glitches on the inputs. These filters are used to reduce noise and transient spikes in order to reduce the likelihood of unintended activation of the reset or the interrupt inputs.

NOTE

The error input pin input of the FCCU is connected to external IRQ pins 0 to 7.

Chapter 6
Failure Rates and FMEDA

6.1 Failure rates

In order to analyze and quantify the effectiveness of the S32V234 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 S32V234. The following methods for deriving the base failure rates of the S32V234 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 S32V234 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 S32V234 enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF)
- 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)) the modules of the S32V234 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 S32V234 can be found in the attached "S32V234 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 S32V234 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 Physical isolation

To maximize the independence of redundant components, these are grouped into spatially separated groups (called 'lakes') and synthesized separately. The groups ensure independence against locally limited faults whereas the synthesis achieves a partial diversity of the logic circuitry.

The redundant modules share a common silicon substrate. A failure of the substrate is typically catastrophic and has to be detected by external system level measures. It is assumed that an external timeout function (watchdog) is continuously monitoring the S32V234 and is capable of detecting this CCF, and will switch the system to a Safe state$_{\text{system}}$ within the FTTI.

The S32V234 device satisfies the standard AECQ100 for latch-up immunity.

7.1.2.2 Environmental conditions

7.1.2.2.1 Temperature

The S32V234 was designed to work within a maximum operational temperature profile (see the *S32V234 Data Sheet*). To cover temperature-related dependent failures, one temperature sensor for supervision is implemented as described in section "Temperature Sensor (TSENS)".

7.1.2.2.2 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.

7.1.2.3 Failures of common signals

7.1.2.3.1 Clock

To cover dependent failures caused by clock issues, modules for supervision are implemented which are described in Clock Monitor Unit (14 x CMU). Major failures in the clock system are also detected by the SWT (Software Watchdog Timer).

7.1.2.3.2 Power supply

To cover dependent failures caused by issues with the power supplies, supervision modules are implemented (see Power Management Controller (PMC)). 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)).

7.1.2.3.3 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 S32V234 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 STCU2 is unintentionally executed during normal operation of the device.
- Any of the configurations for production test are unintentionally executed during normal operation of the device.
- Any JTAGC instruction is executed that causes a system reset or Test Mode Select (TMS) signal is used to sequence the TAP controller state machine.

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

- Usage of peripheral modules not sharing the same PBRIDGE
- 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 *S32V234 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 *S32V234 Data Sheet*) together with the ball number information of the packages as seen in the *S32V234 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 S32V234 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 Modules sharing PBRIDGE

The safety system developer needs to consider how modules are distributed across the different PBRIDGEs. Whenever possible the redundant modules should be connected to a different PBRIDGE.

7.1.3.3 External timeout function

A dependent failure may lead to a state where the S32V234 is not able to signal an internal failure via its FCCU_F*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 S32V234 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 S32V234. As soon as there are no correct communications, the external watchdog function switches the system to Safe state$_{system}$. Thus, either the S32V234 or external watchdog function can transition the system to Safe state$_{system}$. The external watchdog function is required to be sufficiently independent of the S32V234 (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 or FlexRay) and moving the system to Safe state$_{system}$.

7.1.4 Analysis

During the development of the S32V234, the susceptability of the MCU to dependent failures is evaluated by ensuring sufficient independence between on-chip functions and their respective safety mechanisms.

The Dependent Failure Analysis (DFA) is available upon request.

Chapter 8
Acronyms and Abbreviations

8.1 Acronyms and abbreviations

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

Table 8-1. 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 3

A.1 General changes

• Throughout: • Editorial changes and improvements.   • Changed instances of "IRCOSC" to "FIRC".   • Changed instances of "DSPI" to "SPI".   • Changed instances of "XOSC" to "FXOSC".   • Changed instances of "Freescale" to "NXP".
• In "Appendix A - Release Notes":   • Removed the empty "Additional information" section.
• Removed the attached file "S32V234-SM-Allx-words.xlsx".
• Removed references to "Safe mode".

A.2 Preface changes

• In Functional safety standards, changed "ISO 26262-10 Annex A ISO 26262 and microcontrollers" to "ISO 26262-10:2011-2012 Annex A ISO 26262 and microcontrollers". • In Other considerations, changed "The safety system developer" to "The functional safety manager for the developed and deployed system".
• In the Related documentation section: • In the bullet "FMEDA - Inductive analysis enabling customization of system level safety mechanisms...", changed "(SFF and ß-factor $ß_{IC}$)" to "(SFF)".

A.3 MCU Safety Context changes

• In the MCU safety functions section: • In the bullet list of safety functions, changed "Sensor Fusion" to "Data Fusion".
• Cleaned up Figure 2-1. • In the Faults and failures section: • Modified the hierarchy of this section - no content changes.
• In the Continuous reset transitions section, added a cross reference to the Consecutive resets section.

- In the MCU Safe state section, added the "No output" feature to the list.
- In Figure 2-1, added the "Tristated outputs" case.

- In the MCU fault indication time section:
  - In the "Recognition time" bullet, changed "A single full test takes at least 70 μs" to "Refer to the "Self-test" section in the Analog-to-Digital Converter (SAR-ADC) chapter in the Reference Manual".

- In the MCU fault indication time section:
  - In the "FCCU configured as "slow switching mode"" bullet:
    - Changed the error out frequency from 43 Hz to 434 Hz.
    - Changed the maximum indication delay from 11.6 ms to 1.15 ms.

A.4 Functional Safety Concept changes

- In the General concept section:
  - Editorial update.
- In the ECC for storage section:
  - In the "A53" row, changed "L1 I-cache dirty" to "L1 D-cache dirty".

- In the General concept section:
  - Updated the existing table Table 3-2.
  - Added new table Table 3-3.
- In the ECC for storage section:
  - Updated Table 3-4 :
    - Split the "A53" rows into "A53" and "A53-cluster".
    - Removed the "APU DMEM" row.
    - Renamed the "APU IMEM" row to "APU SMEM".
    - Added the "VIU" row.
- In the Communication controllers section:
  - Added LINFlexD to the list of communication controllers that do not contain special safety mechanisms.
- Added the Triple-voted flops (TVF) section.

- In Table 3-1 :
  - In the "Processing units (ARM Cortex CA53 Core)" row:
    - Changed "(optional) Spatial hardware redundancy (multiple cores) supported by software (reciprocal comparison)" to "Spatial hardware redundancy (separated clusters) supporting optional software redundancy (e.g. reciprocal comparison)".
  - In the "Image processing units (APEX2)" row:
    - Changed "Software based temporal redundant processing (video frames)" to be optional.
    - Removed "Software diversified redundancy (processing mirrored images)".
  - In the "Interrupt" row:
    - Removed "Multiple interrupt destinations supervised by software".
  - In the "DMA" row:
    - Changed "Data block signature (CRC...) hardware supported by software" to "Data block signature (CRC) when moving data between SRAM and DRAM".
  - In the "Volatile memories - external (DDR DRAM...)" row:
    - Removed "Temporal and spatial redundancy of vision frames (software measure)".
    - Added "ECC includes address protection".
  - In the "Volatile memories - integrated (SRAM, cache...)" row:
    - Changed "Some memories with address protection" to "Address protection for selected memories".
  - In the "Non-volatile memories - external (Flash)" row:
    - Changed "Block code (software measure supported by hardware 128-bit AES CTR code)" to "Block code (software measure supported by hardware 128-bit AES)".
- In the DMA lockstep section:
  - Moved the sentence/paragraph "A delay of two clock cycles..." to the end of the subsection.
  - In the above sentence, changed the phrase "checker channel" to "checker channel of eDMA".
  - In the above sentence, added the phrase "(delayed lock-step)" to the end of the sentence.
- In Table 3-4 :

• Added the footnote "Faults in the ECC logic itself are also detected during run-time (EDC-after-ECC)" to HPSMI and MMDC_0. • Added the footnote "ECC is not supported in cut 1" to MMDC_1. • Updated the rest of the MMDC_1 row to reflect the "ECC is supported" case. • In the ECC failure handling section: • Added the following to the end of the section: "Correctable errors should be transparently corrected without reporting to not disturb the application. This means that correctable errors for type c, d and e are typically disabled in FCCU." • In the All-X words and ECC section: • Moved the "All-X words and ECC" section from Chapter 3 to Chapter 5.
• In the DMA lockstep section: • In the first two paragraphs, changed "delayed lockstep" to "lockstep".
• In Table 3-1 :   • In the "Timers" row, changed "Two Instantiations of Periodic Interrupt Timer (PIT) with 8 channels" to "Two Instantiations of Periodic Interrupt Timer: PIT0 supports 6 channels, PIT1 supports 4 channels".   • In the "Communications" rows, changed instances of "FD-CAN" to "CAN FD". • In Table 3-3 :   • In the "Measure improving independence" row, changed "xRDC" to "XRDC". • In Table 3-4 :   • In footnote 1, specified that "Faults in the ECC logic itself" applies to MMDC_0. • In the Power section: • Removed the reference to the "Power Control Unit (MC_PCU)" chapter. • In the Built-In Self Tests (BIST) section: • In the last paragraph, added that "External flash memory (accessible through QSPI)" is also not covered by MBIST. • In the Built-In Self Tests (BIST) section: • Removed the "Logical BIST (LBIST)" subsection. • In the Operational interference protection section: • In the paragraph "A local protection within the Shared Memory Interconnect (HPSMI) exists which protects from unintentional memory overwriting", changed "JPEG" to "JPEGD (JPEG Decoder)" in the module list.

A.5 Hardware Requirements changes

• In the Hardware requirements on system level section: • Corrected instance of FCCU_F[n] to FCCU_F*n*.
• In the Error Out Monitor (ERRM) section: • Changed "FCCU_F0, and optionally FCCU_F1" to "FCCU_F0 and/or FCCU_F1".
• In the Power Supply Monitor (PSM) section: • In Assumption SM_087, removed the phrase "where no supervision is provided on the MCU".
• In Power Supply Monitor (PSM) :   • Removed the sentence: "If the S32V234 power supply can be designed to avoid any potential of over-voltage, the external voltage monitoring can be excluded from the system design."

A.6 Software Requirements changes

• In the Power Management Controller (PMC) section: • In Assumption SM_144, moved "(per the default configured in the PMC fuses)" to be a separate sentence following the assumption. • In the IRCOSC Initial checks and configurations section:

  - Separated the second portion of Assumption SM_703 to be an "Implementation hint".
  - Removed the cross reference to "the Frequency meter section in the "Clock Monitor Unit (CMU)" chapter of the S32V234 Reference Manual".
- In the External Flash section:
  - Editorial updates (changed "FLASH" and "Flash" to "flash").
  - Changed SM_909 from "Assumption" to "Assumption under certain preconditions".
  - Changed SM_910 from "Assumption" to "Assumption under certain preconditions".
- In the Freedom from interference section:
  - In Assumption SM_092, removed the condition "(if necessary)".
- In the Temperature Sensor (TSENS) section:
  - In the "FCCU failure generation" bullet, changed "low and high temperature points" to "high temperature point".

- In the Power Management Controller (PMC) section:
  - Updated Assumption SM_085.
  - Replaced the "PMC monitored supplies" table, Table 5-1.
- In the Phase-locked loop (5 x PLL) Initial checks and configurations section, changed the following:
  - In Assumption SM_078, changed "system clock of the S32V234" to "source clock for the PLLs".
  - Removed Assumption SM_079 and its accompanying Rationale.
  - Removed the Implementation hint: "Either before or during initialization..."

- In the CMU Initial checks and configurations section:
  - Changed Assumption SM_902 to a standard paragraph.
  - Added a cross reference to the "Software Watchdog Timer" section.
- In the External Oscillator (FXOSC) section:
  - Removed the "Runtime checks" subsection.
- In the External Flash section:
  - Added the Recommendation: The CRC feature of the FastDMA is intended to perform the array integrity check at startup.
- In the Volatile memory section:
  - Replaced the sentence "Memories using ECC with address encoding must be initialized by software" with "Some memories using ECC with address encoding must be initialized by software. This is applicable for system SRAM (HPSMI, CPU memories) and the physical portion of DRAM which is protected by ECC."
  - Removed Assumption SM_915.
  - Removed the Implementation hint.
- In the DRAM Controller ECC (MMDC_ECC) section:
  - Added the introductory text "The DRAM controller embedds ECC data within the normal data to the DRAM..."
  - Removed Assumption SM_949.
- In the JTAG Controller (JTAGC/DEBUG) section:
  - Changed Assumption SM_906 to an Implementation hint.
  - Changed Assumption SM_907 to a standard paragraph.
- In the Core self-test section:
  - In Assumption SM_959, changed "within the FTTI on each core" to "within the FTTI on each A53 core."
- In the Redundant code execution section:
  - Editorial change.
- Moved section Cortex M4 to section Processing modules and added new subsections to it.
- In the Software Watchdog Timer section:
  - In Assumption SM_202, removed the sentence "Detection latency shall be smaller than the FTTI."
- In the CRC Runtime checks section:
  - In Assumption SM_070, added the condition "if a wrong setting is not detected by other means."
  - Removed the Implementation hint.
  - Removed the paragraph beginning with "At run time, the value calculated by the CRC module..."
- In the FCCU Runtime checks section:
  - Removed Assumption SM_932.
- In the Error reporting path tests section:
  - Added the content at the end of the section beginning with "Software can program two patterns into RAM..."
- In the Built-in Hardware Self-Tests (BIST) section:
  - Changed "STCU2_MBSLSW, STCU2_MBSMSW and STCU2_MBSHSW" to "STCU2_MBSSWn".
  - Changed "STCU2_MBELSW, STCU2_MBEMSW and STCU2_MBEHSW" to "STCU2_MBESWn".
  - Removed Assumption SM_109.
- Switched the order of the Video Encoder H.264 and Video Decoder H.264 sections.
- In the MIPI-CSI2 section:

    - Removed Assumption SM_926.
    - Removed Assumption SM_927.
- In the ISP section:
    - In Assumption SM_921, changed "FastDMA shall be used" to "FastDMA should be used".
    - Changed Assumption SM_921 to be an Implementation hint.
- In the Communications section:
    - Moved Assumption SM_051 to the "Fault-tolerant communication protocol" subsection.
    - Removed the remaining content from this section (not including subsections).
- In the Fault-tolerant communication protocol section:
    - Added PCIe, Ethernet, and LFAST/SIPI to the list of modules for which a software interface with the hardware channel is required if used for functional safety.
    - Added Assumption SM_051.
- In the Ethernet assumptions section:
    - Removed Assumption SM_912.
- In Cortex A53 core cluster :
    - Added Figure 5-1.
- In Video Decoder H.264 :
    - Added Figure 5-2.
- In the Digital I/O section:
    - Changed title from "Digital inputs" to "Digital I/O".
    - Changed "input" to "input or output".
    - Editorial change.
- In the PLLDIG Initial checks and configurations section:
    - Changed "deactivated" to "powered down".
    - In Assumption SM_078, specified the clock source to be FXOSC.
- In the Internal RC Oscillator section:
    - Changed "CMU 7" to "CMU_7".
- In the Local clocks section:
    - In the bullet list, changed "MIPI_CSI2 (for test only)" to "MIPI_CSI (for test only)".
    - Specified that "Faults within these modules are expected to be covered by the Fault-tolerant communication protocol" refers to "LFAST or PCIe".
- In the Volatile memory section:
    - Removed the subsection "System SRAM Controller (HPSMI)".
    - Moved the All-X words and ECC section from Chapter 3 to Chapter 5.
- In the JTAG Controller (JTAGC/DEBUG) section:
    - In the Assumption SM_048 list of modules:
        - Changed "Deserial Serial Peripheral Interface (SPI)" to "Serial Peripheral Interface (SPI)".
        - Removed "FlexRAY".
        - Added "eDMA, ENET, SIPI".
- In the Interrupt Controller (GIC, NVIC) section:
    - Changed this subsection's title from "Interrupt Controller (GIC)" to "Interrupt Controller (GIC, NVIC)".
    - Removed Assumption SM_098 and the Implementation hint.
- In the Peripheral lake eDMA transfers section:
    - Changed instances of "eDMA Channel Mux" to "DMA Channel Mux".
    - Changed "PIT_0" to "PIT0".
- In the Non-replicated eDMA transfers section:
    - Changed instances of "eDMA Channel Mux" to "DMA Channel Mux".
- In the Redundancy Control Checking Unit section:
    - Changed "The RCCUs are automatically enabled" to "The RCCUs are always enabled".
- In the Cortex A53 core cluster section:
    - Changed the title of this subsection from "A53 core cluster" to "Cortex A53 core cluster".
- In the Freedom from interference section:
    - Changed "Both cluster are physically separated and interference is low due to the structural separation with two separated L2 caches and possibility to exclude cache coherency" to "Both clusters are physically separated and interference is low due to the structural separation with layout restrictions, two separated L2 caches, and the possibility to exclude cache coherency".
- In the Cortex M4 section:
    - Changed the title of this subsection from "ARM Cortex M4" to "Cortex M4".
- In the Core self-test section:
    - Changed the title of this subsection from "ARM Core self-test" to "Core self-test".

- In the Black-channel section:
  - Changed the title of this subsection from "ARM Black-channel" to "Black-channel".
- In the Fault Collection and Control Unit (FCCU) section:
  - In the bullet sublist of "Internal reactions", added "NMI (Non-maskable Interrupt)".
- In the Built-in Hardware Self-Tests (BIST) section:
  - Removed "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)."
  - Replaced the paragraph "If there is an LBIST failure..."
  - Moved "Assumption under certain conditions: [SM_914]", and the short paragraph following it, to the end of the Scheduling subsection.
- In the Scheduling section:
  - Changed this subsection's title from "MBIST" to "Scheduling".
  - Changed "SRAM BIST (MBIST) runs during initialization..." to "Logic BIST (LBIST) and SRAM BIST (MBIST) runs during initialization..."
  - Removed the reference "see Self Test Control Unit (STCU2)".
  - Added the note "In principle LBIST can be run at any time, but the MCU will execute a reset after LBIST completes" from the "LBIST" subsection, then removed that subsection.
  - Added all content from the "ADC BIST" section, then removed that section.
  - Added all content from the "PMC LVD/HVD tests" section, then removed that section.
  - Moved "Assumption under certain conditions: [SM_914]", and the short paragraph following it, from Built-in Hardware Self-Tests (BIST) to the end of this subsection.
- In the Wake-Up Unit (WKPU) / External NMI section:
  - Changed Assumption SM_126 from "If external NMI is used as a safety mechanism, especially if waking up within a certain timespan or at all is considered safety-relevant, it is required..." to "If external NMI is used as a safety mechanism, it is required...".
- In the Processing units for vision section:
  - Changed the title (was "Processing units vision").
  - Added the three paragraphs beginning with "Many processing steps in vision processing modules..."
- In the Video Decoder H.264 section:
  - Removed the sentences "These errors are either not seen by the driver or the picture may be massively corrupted so that the driver does not rely on it. For frozen images, please refer to section the 2D-ACE/DEC200 section."
- In the VIULite section:
  - In Assumption SM_947, changed the word "frequently" to "periodically".
- In the Peripheral section:
  - Removed the "ARM Cortex M4" subsection.
- In the Fault-tolerant communication protocol section:
  - In the second paragraph, changed the phrase "is required" to "may be required".
  - Changed "FlexRay, 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" to "Communication interfaces do not have specific functional safety mechanisms for single-point faults other than what is included in their protocol specifications or ECC protection of SRAM where applicable".
- In the System Integration Unit Lite2 (SIUL2) section:
  - Removed the paragraph "The integrity of functional safety-relevant periphery..."

- In the All-X words and ECC section:
  - Changed "ALL-x" and "All-x" to "All-X".
- In the Test mode section:
  - Changed the last sentence in the "Implementation hint" to be a separate paragraph.
  - Added this sentence to the end of the section: "FIRC clock-related testmode activation is intended to be covered by the frequency meter function of CMU_0, as described in section Runtime checks."
- In the Interrupt Controller (GIC, NVIC) section:
  - Changed occurrences of "(GIC)" to "Interrupt controller".
- In the Periodic low latency IRQs section:
  - Changed "Interrupt Control Monitor (INTCM)" to "Interrupt Monitor (INTM)".
- In the Cortex M4 section:
  - Changed "detection of faults on the core cluster" to "detection of faults".
- In the Fault Collection and Control Unit (FCCU) section:
  - Changed "8000 FIRC cycles" to "8000 FIRC3_CLK cycles".

- In Power Management Controller (PMC), added the footnote "There are internal connections from the PMC to the ADC" to the "ADC" column heading.
- In the 2D-ACE/DEC200 section:
  - Added the following at the end of the section: "The HW cursor layer of the 2D-ACE should not be used for applications which display safety relevant information on the screen."

- In the Clock Monitor Unit (14 x CMU) section:
  - In Assumption SM_080, moved the sentence "The FCCU's default condition does not manage the CMU faults, so it must be configured accordingly" to appear after the assumption as a separate paragraph.
- In the Glitch filter section:
  - Changed "A selectable (WKPU_NCR[NFE0]) analog glitch filter is implemented on the NMI-input" to "A selectable (WKPU_NCR[NFE0]/WKPU_NCR[NFE1]) analog glitch filter is implemented on the NMI input."

- In JTAG Controller (JTAGC/DEBUG), removed the following: "If the FCCU recognizes erroneous activation of debug mode. You can enable a feature of the FCCU that resets the MCU if it detects that JTAG has been connected to the device, this is a safety and security feature. To provide notification that the device accidentally went into debug mode while in production. It can also be used to prevent hacking."
- In Power Management Controller (PMC), changed "Over voltage of any 3.3 V supply shall be monitored externally as described in Power Supply Monitor (PSM)" to "Over voltage of supply shall be monitored externally as described in Power Supply Monitor (PSM)".

A.7 Failure Rates and FMEDA changes

- In the FMEDA section:
  - In the "FMEDA" bullet, removed "and IEC 61508 (SFF and ß-factor ßIC)".

- In the Module classification section:
  - In the first paragraph, changed "IEC 61508 (Safe Failure Fraction (SFF) and $ß_{IC}$ factor)" to "IEC 61508 (Safe Failure Fraction (SFF))".

A.8 Dependent Failures changes

- In the Analysis section:
  - Changed this section's title from "$β_{IC}$ considerations" to "Analysis".
  - Removed the paragraph "One method to do this..."
  - Changed "The FMEDA includes the ßIC calculations and is available upon request" to "The Dependent Failure Analysis (DFA) is available upon request".

A.9 Acronyms and Abbreviations changes

- No substantial content changes

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number S32V234SM
Revision 3, 10/2017