Safety Manual for MPC5744P

Devices Supported: MPC574xP

Document Number: MPC5744PSM
Rev. 4, 02/2018

Contents

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

Chapter 1
Preface

Chapter 2
MCU safety context

Section number	Title	Page

Chapter 3
MCU Safety Concept

Chapter 4
Hardware Requirements

Chapter 5
Software requirements

Section number	Title	Page

Section number	Title	Page

Section number	Title	Page

Chapter 6
Failure Rates and FMEDA

Chapter 7
Dependent Failures

Section number	Title	Page

Chapter 8
Additional Information

Chapter 9
Acronyms and Abbreviations

Chapter 1
Preface

1.1 Overview

This document discusses requirements for the integration and use of the MPC5744P 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 MPC5744P, 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 MPC5744P is developed according to ISO 26262 and has an integrated safety concept.

1.2 Safety manual assumptions

During the development of the MPC5744P, 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 $\text{state}_{\text{system}}$) when the safe state of the MCU (Safe $\text{state}_{\text{MCU}}$) 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 MPC5744P.

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 MPC5744P 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 MPC5744P. 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 MPC5744P 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 MPC5744P 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 MPC5744P 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 MPC5744P into safety-related systems, the following documentation is available:

- Reference Manual (MPC5744PRM) - Describes the MPC5744P functionality
- Data Sheet (MPC5744PDS) - Describes the MPC5744P operating conditions
- Safety Manual (MPC5744PSM) - Describes the MPC5744P safety concept and possible safety mechanisms (integrated in MPC5744P, system level hardware or system level software), as well as measures to reduce dependent failures
- FMEDA - Inductive analysis enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and ß-factor $ß_{IC}$)
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The Reference Manual, Data Sheet and Safety Manual are available for download on the MPC5744P product page. The FMEDA and FMEDA report are available upon request. The MPC5744P 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 MPC5744P, the following information should be considered:

- The MPC5744P is handled in accordance with JEDEC standards J-STD-020 and J-STD-033.
- The operating conditions given in the MPC5744P Data Sheet.

- If applicable, any published MPC5744P errata.
- The recommended production conditions given in the MPC5744P quality agreement.
- The functional safety manager for the developed and deployed system is required to report all field failures of the MPC5744P 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. To locate any published errata or documentation updates, visit the MPC5744P product page

Chapter 2
MCU safety context

2.1 Target applications

As a SafeAssure solution, the MPC5744P microcontroller targets applications requiring a high Automotive Safety Integrity Level (ASIL), especially:

- Chassis applications
- Electrical Stability Control (ESC)
- Higher-end Electrical Power Steering (EPS)
- Airbag and sensor fusion applications
- Radar applications

A typical aspect of ESC and EPS is the presence of an advanced electrical motor control periphery with special enhancements in the area of pulse width modulations, highly flexible timers, and functional safety. All devices in this family are built around a safety concept based on delayed lock step, targeting an ISO26262 ASIL-D integrity level.

2.2 Safety integrity level

The MPC5744P is designed to be used in automotive, or industrial, applications which need to fulfill functional safety requirements as defined by functional safety integrity levels (for example, ASIL D of ISO 26262 or SIL 3 of IEC 61508). The MPC5744P 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 MPC5744P is considered a Safety Element out of Context (SEooC) development.

The MPC5744P 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 MPC5744P, no specific safety function can be specified. Therefore, during the SEooC development of the MPC5744P, 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:

- **Software Execution Function (Application Independent)**: Read instructions out of the MPC5744P flash memory, buffer these within instruction cache, execute instructions, read data from the MPC5744P System SRAM or flash memory, buffer these in data cache, process data and write result data into MPC5744P System SRAM. **Functional safety of the Software Execution Function is primarily achieved by safety mechanisms integrated on the MPC5744P**.

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

- **Input / Output Functions (Application dependent)**: Input / Output functions of the MPC5744P 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 MPC5744P 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 $\text{state}_{\text{system}}$, whereas a safe state of the MPC5744P is named Safe $\text{state}_{\text{MCU}}$. A Safe $\text{state}_{\text{system}}$ is an operating mode without an unreasonable probability of occurrence of physical injury or damage to the health of any persons. A Safe $\text{state}_{\text{system}}$ 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 $\text{state}_{\text{system}}$) when the safe state of the MCU (Safe $\text{state}_{\text{MCU}}$) is entered. [end]

2.4.1 MCU Safe state

The safe states (Safe $\text{state}_{\text{MCU}}$) of the MPC5744P 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)

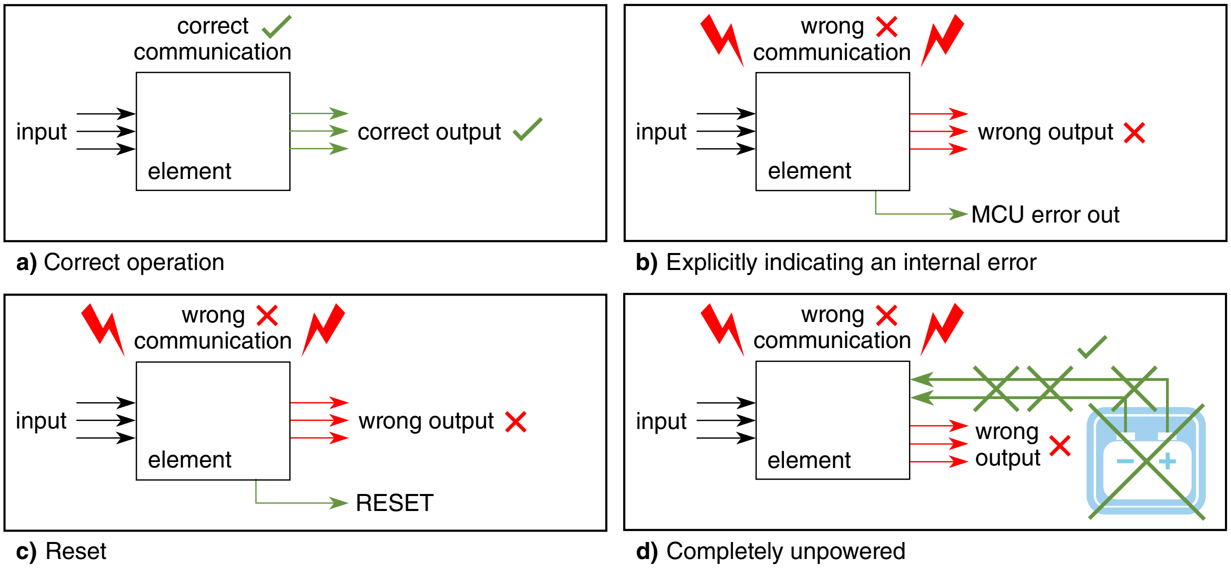


Figure 2-1. Safe state$_{MCU}$ of MPC5744P

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_F[0] or FCCU_F[1]). [end]

Implementation hint: If the MPC5744P signals an internal failure via its error out signals (FCCU_F[*n*]), the surrounding subsystem shall no longer use the MPC5744P 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 MPC5744P. Depending on the configuration, the system may disable, or reset, the MPC5744P 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]

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 MPC5744P.
- **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.

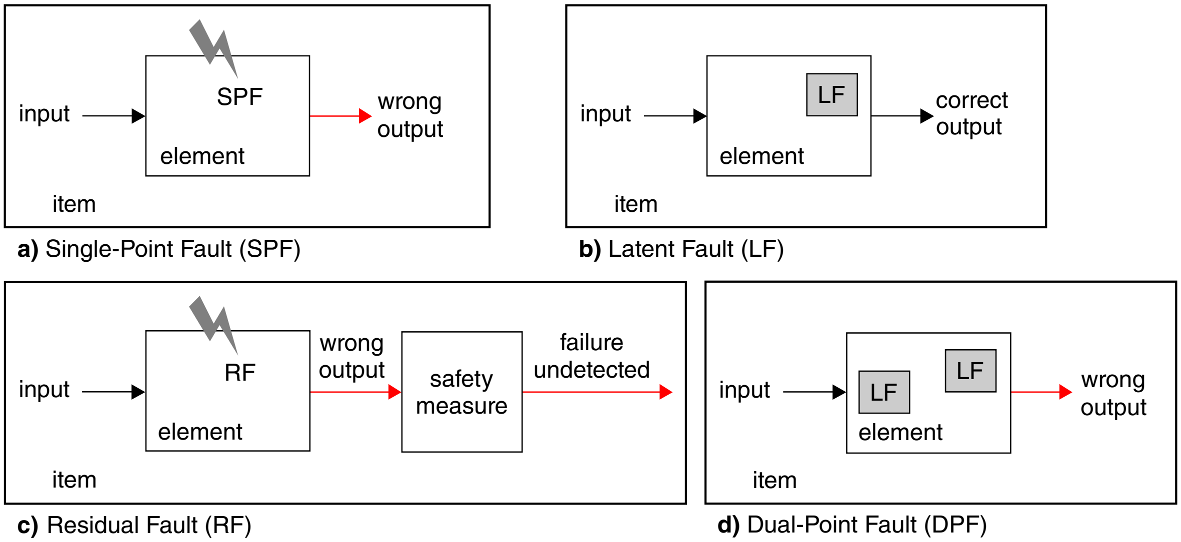


Figure 2-2. Faults

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

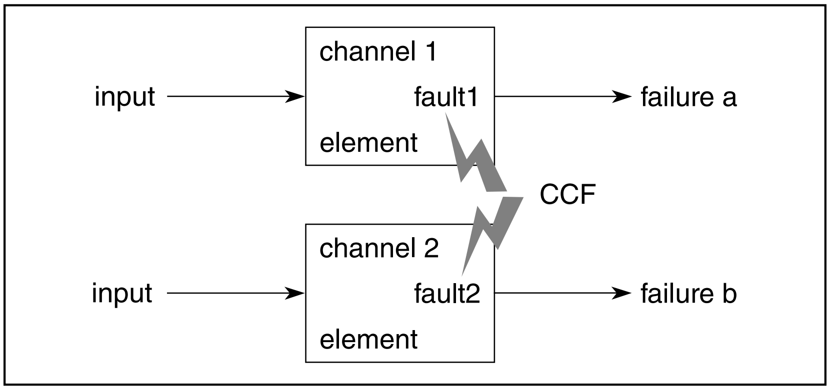


Figure 2-3. Common Cause Failures

- **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.

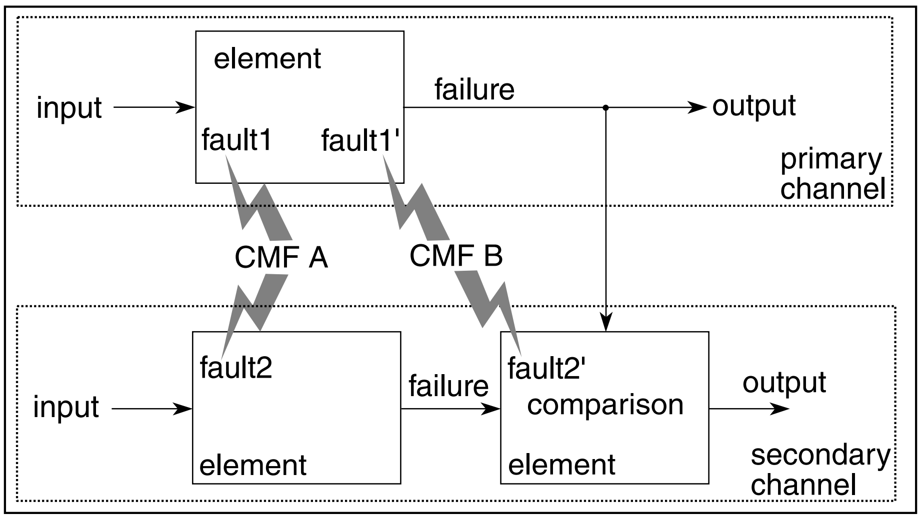


Figure 2-4. Common Mode failures

- **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).

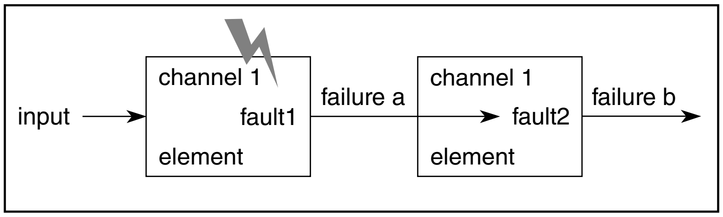


Figure 2-5. Cascading failures

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.

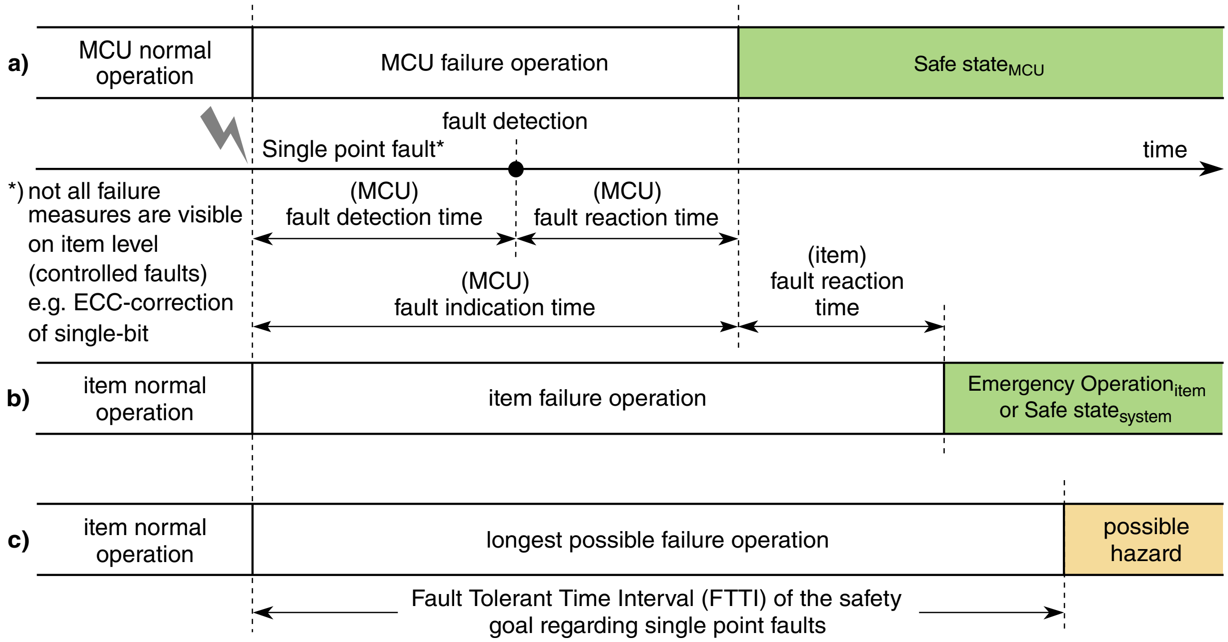


Figure 2-6. Fault tolerant time interval for single point faults

Fault indication time is the time from the occurrence of a fault to when the MPC5744P is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the MPC5744P 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. A single full test takes at least 70 µs.

- 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 Internal RC Oscillator (IRCOSC) clock cycles (nominal frequency of 16 MHz).
  - **External indication time** to notify an observer about a failure external to the MPC5744P. 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 IRCOSC frequency of 16 MHz, the error out frequency is 61Hz. Therefore, the maximum indication delay is 8 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, XBAR contention, register saving, and so on).

The sum of the MPC5744P 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.

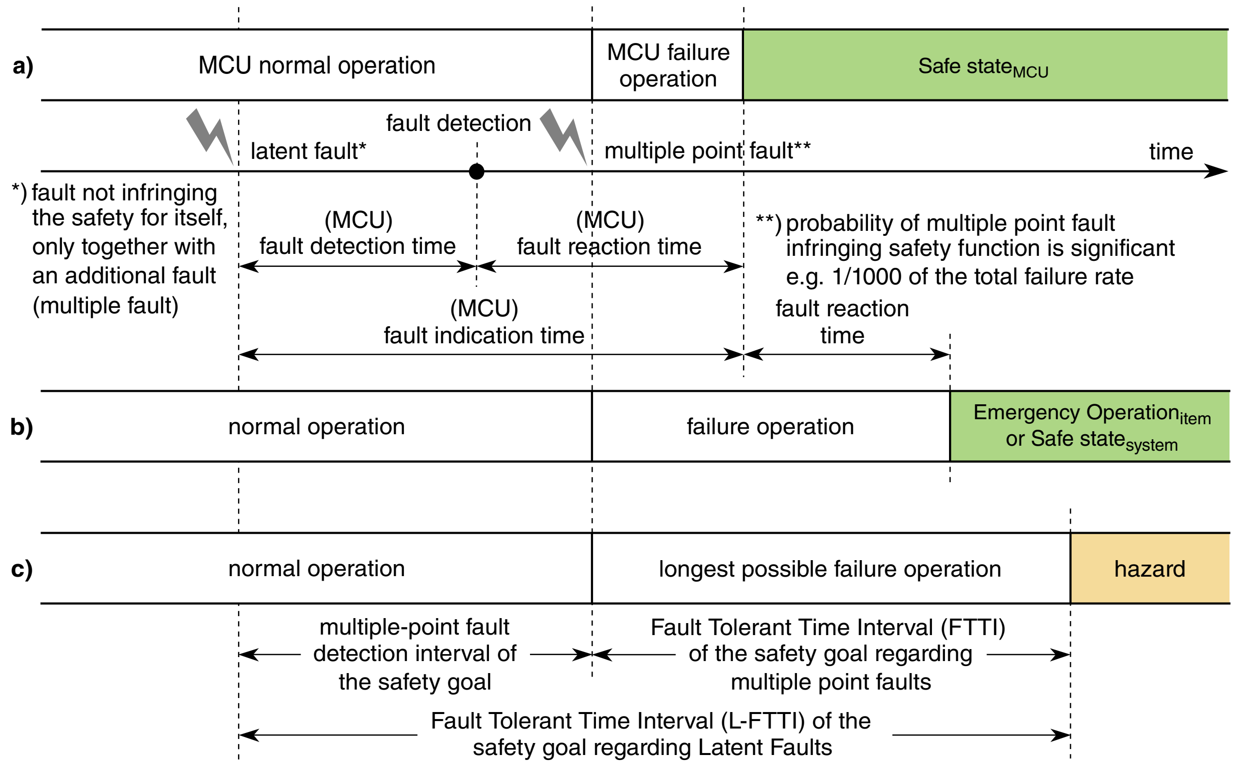


Figure 2-7. Fault Tolerant Time Interval for latent faults

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 MPC5744P 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 MPC5744P 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 MPC5744P.

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 MPC5744P. The FCCU provides two error output signals (FCCU_F[0] and FCCU_F[1]) used for external failure indication.

Different protocols for the error output pins are supported:

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

After power-on reset, the 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_F[0] = 0 and FCCU_F[1] = 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

Figure 3-1 is a top-level diagram showing the functional organization of the MPC5744P.

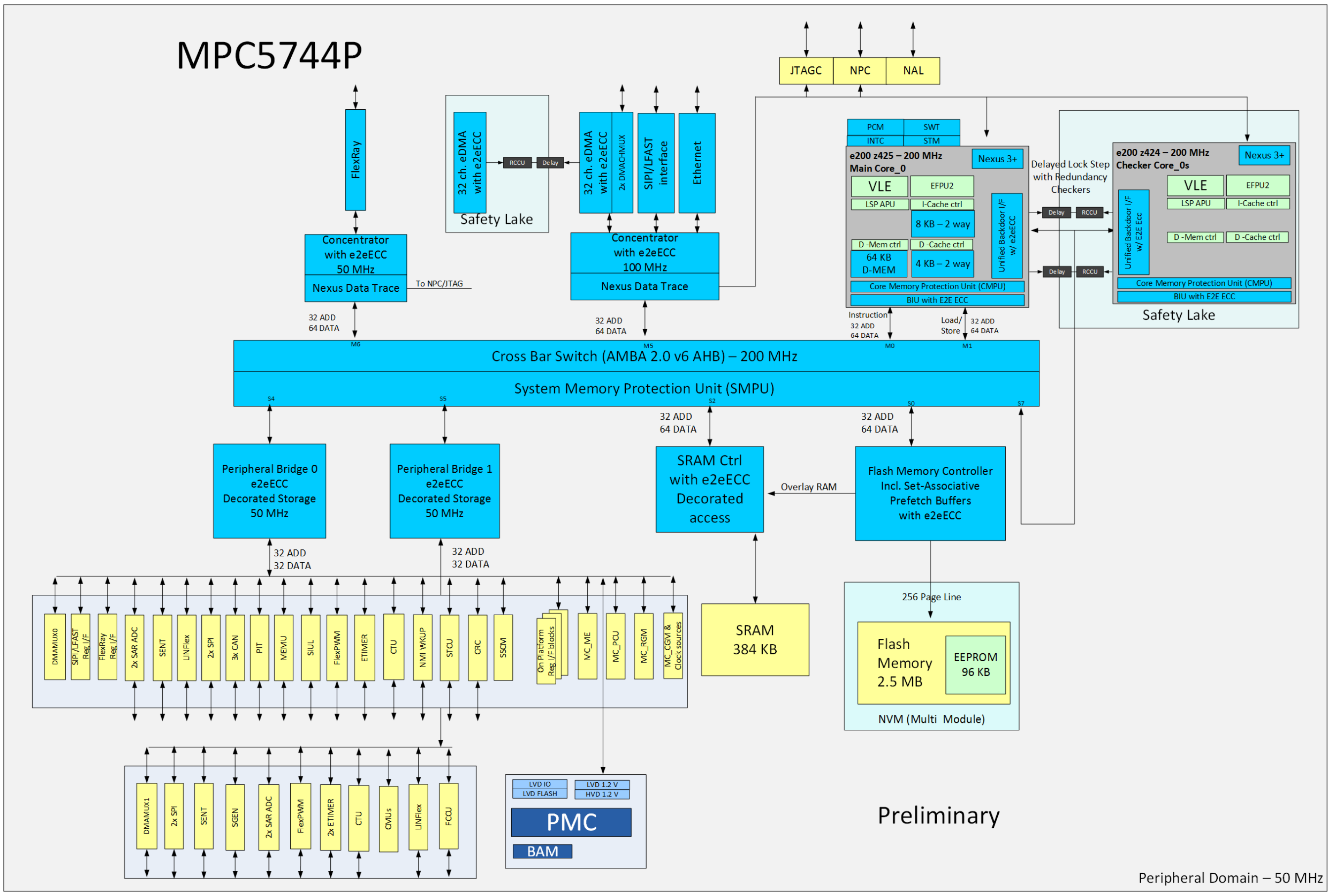


Figure 3-1. MPC5744P block diagram

The MPC5744P has an integrated safety concept targeting safety-related systems requiring high safety integrity levels. In general, safety integrity is achieved in the following ways:

- For the Safety Core and its closely related periphery, safety is ensured by a delayed lockstep approach (see section Dual-core lockstep)
- The safety of storage and of the data path to storage and periphery is ensured by End-to-End ECC (e2eECC) with address encoding and selected additional safety measures for individual modules. For the periphery, end-to-end ECC protection ends at the I/O bridges (see section ECC)
- Clock and power, generation and distribution, are supervised by dedicated monitors (see section Clock and power monitoring)
- The safety of the periphery is ensured by application-level measures (such as connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on). Hardware supports this application-level redundancy by providing redundant I/O modules connected to different peripheral bridges (PBRIDGEs) to maximize the independence between the monitored and monitoring resources (see sections I/O peripherals and Communication controllers)
- MBISTs and LBISTs are provided to avoid the accumulation of latent faults in the functional logic as well as in the safety mechanisms (see section BIST during boot). Dedicated mechanisms are provided to check the availability of safety mechanisms and the functionality of each error reaction path (such as by fake fault injection)
- The Fault Collection and Control Unit is responsible for collecting and reacting to failure notifications (see section FCCU and failure monitoring)
- For error events including ECC corrections and detections: The MEMU is responsible for collecting and reporting to the FCCU error events in system memories and flash memory as well as e2eECC errors caused by the XBAR, RAM controller, or flash memory controller (see section Memory Error Management Unit (MEMU))
- Common Cause Failures (CCFs) are dealt with by a set of measures for both control and avoidance of CCFs spanning system-level approaches (such as temperature and nonfunctional signal monitoring) and back-end techniques (such as isolated silicon areas and routing constraints) (see section Common cause failure measures)
- Operational interference protection is ensured via a hierarchical memory protection schema allowing concurrent execution of software with different (lower) ASIL (see section Operational interference protection)

3.2 Dual-core lockstep

The MPC5744P duplicates its safety-relevant processing elements and compares their operation in Lockstep mode (LSM). This Safety Core consists of two cores, Checker Core_0 and Master Core_0, and as far as software is concerned they behave as one core. Main Core_0 is the main execution core of the pair, where Checker Core_0 follows the execution of the Master core in lockstep.

The processing elements which are replicated contain:

- Core
- Cache control
- Local memory control
- Core Memory Protection Unit (CMPU)
- Core System Bus Interface, including E2E ECC logic
- eDMA controller

Together each set of replicated elements forms a channel (for example, the Main channel and the Checker channel). Equivalent operation of replicated resources is supervised by comparators on all functional signals leaving the channels for the rest of the MCU. Any operational deviations between the supervised signals will cause the FCCU to be notified of the discrepancy.

The Checker Core does not have a direct connection to the XBAR. All of the outputs of Checker Core_0 that target the XBAR (as well as any other non-duplicated resource, like local memories) will end in an RCCU for verification, and all the inputs to Checker Core_0 from the XBAR will be split off from the Main Core_0 XBAR inputs.

An abstract view of the implementation including delays and RCCUs is shown in Figure 3-2, but it does not show the local memory.

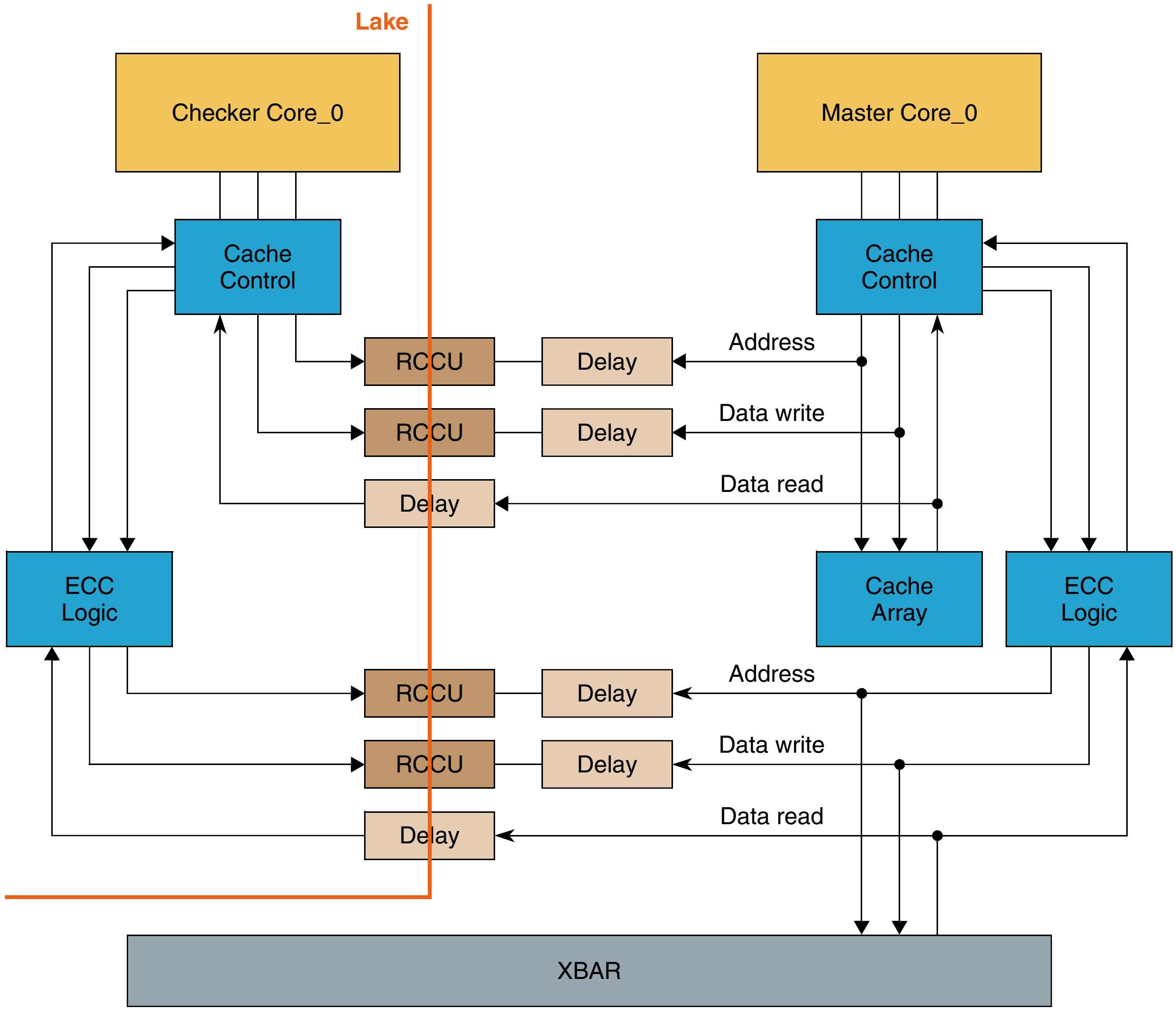


Figure 3-2. Dual core concept

A delay of two clock cycles exists between the Master and Checker channel to reduce their susceptibility to CCFs caused by power and clock disturbances. Outputs of the Master core (such as XBAR addresses) are registered and delayed before being forwarded to the RCCUs while input signals (such as XBAR read data) are registered and delayed before being fed to the Checker channel.

3.2.1 Redundancy Control and Checker Units (RCCUs)

The 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 core against the output of the Master core and issues an alarm if they do not match. In case of a compare mismatch the fault stays stable until it is explicitly cleared.

3.3 ECC

Error correcting codes are used for end-to-end protection from cores to system storage as well as for individual protection of peripheral RAMs.

3.3.1 End-to-End protection on data path

Connections between XBAR masters and slaves (clients) are denoted as data paths. Data corruption on all data paths between the core and any client is detected via two main safety mechanisms: data from the masters is encoded using Error Correcting Code (ECC), which is implemented with a Single-Error Correction, Double-Error Detection (SECDED) code with a Hamming distance of 4 and includes coverage of addressing information. Control signals and address decoding are monitored to verify the data reaches all of the intended clients, from all possible connections to these clients and the intended operation is performed on the target address. Figure 3-3 illustrates the overall ECC schema.

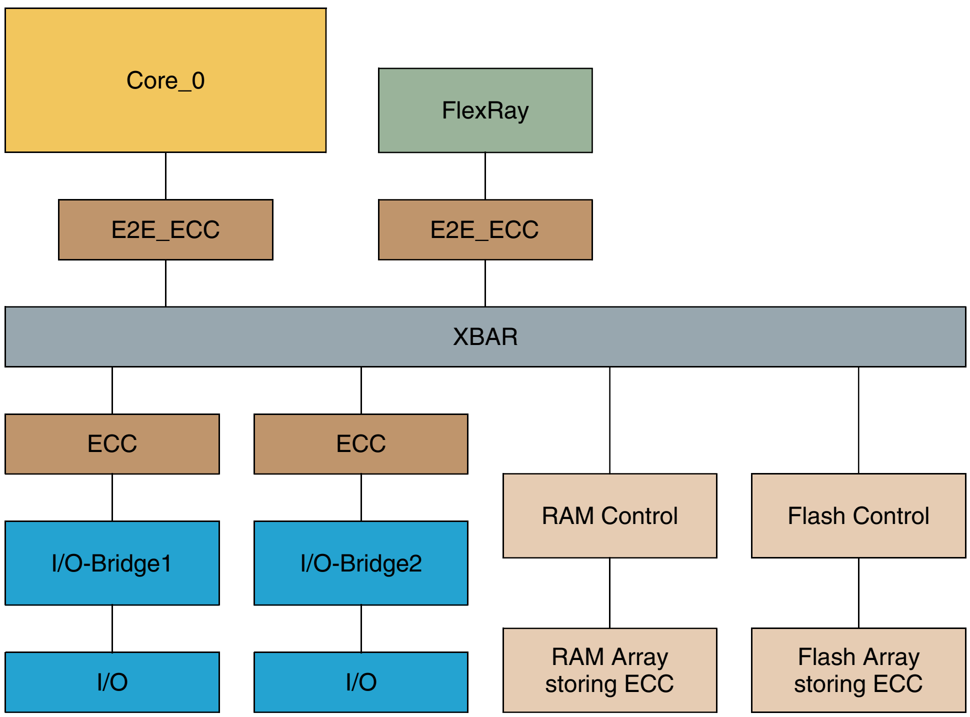


Figure 3-3. General view of e2eECC

NOTE

> Specific implementations for the MPC5744P vary depending on the special requirements of RAM and flash memory concerning ECC handling, as well as for caches, local RAM, tag memories of the cores and DMA RAM.

ECC bits are generated on writes by XBAR masters (including, but not limited to the core) and checked on reads. The ECC correction bits are stored alongside the data in flash memory and RAM so, in principle, no ECC logic is necessary at the memories themselves. For this reason the ECC schema is referred to as End-to-End ECC (e2eECC) in the following sections. For XBAR slaves, other than memories, new ECC logic is added as these clients cannot store or produce the ECC correction bits. This resolves the problem where ECC needs to be calculated in real time before entering or exiting the ECC-protected data path. This is particularly true with peripherals connected to the I/O bridges. This setup is considered sufficient to fulfill safety requirements because the data path not protected by ECC, which is downstream from the I/O bridges, is replicated and is used redundantly by the application (see section I/O peripherals).

The e2eECC schema provides high detection capabilities against failures affecting the data content of the transaction. The inclusion of the target address in the computation of the redundancy bits (8 ECC bits) does allow the partial detection of addressing faults as well. To reach the desired integrity level, additional dedicated safety mechanisms are implemented in the data path particularly to:

- Improve the detection capability over addressing failures (no/multiple/wrong address selected), considering faults affecting address transmission (from master to client) as well as the decoding of the address;
- Provide coverage for control failures affecting, for example, the type (read vs. write) or size of a transaction.

Though safety mechanisms protecting the XBAR, the RAM controller, or the flash memory controller are different, they are all based on the feedback of address and control information from the target to the source of the transaction, which is responsible for checking for consistency with respect to the intended transaction. Depending on the portion of the data path covered by the specific safety mechanism, the source can be an XBAR master port rather than the XBAR interface of the RAM or Flash Memory Controllers; the target is respectively an XBAR slave port, the RAM array, or the flash memory module. See the separate *MPC5744P Reference Manual* chapters dedicated to the Crossbar Switch (XBAR), Flash Memory Controller (PFLASH), and RAM Controller (PRAMC) for further details.

NOTE

The address and control feedback mechanism also covers caches, local RAM, tag memories of the cores and DMA RAM.

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 are protected by EDC (Error Detection Code). The list showing the implementation of RAMs with ECC (including address protection) is shown in the following table.

Table 3-1. ECC RAM implementations

Module	Memory	Memory column muxing factor	ECC	Address in ECC
Main Core_0	I-Mem	8	SEC/DED	Y
	D-Mem	8	SEC/DED	Y
	I-Cache	4	EDC	Y
	D-Cache	4	EDC	Y
	ITAG	4	SEC/DED	Y
	DTAG	4	SEC/DED	Y
System RAM	System RAM	16	SEC/DED	Y
DMA	DMA	4	SEC/DED	Y
FlexRay	DRAM	4	SEC/DED	N
	LRAM	4	EDC	N
CAN	CAN	4	SEC/DED	N
Ethernet	ENET	4	No	N

Some memories, particularly system storage, use an ECC computed over data and address to detect data and addressing faults (no/wrong/multiple selection). In addition, some of these memories include dedicated measures against addressing and control faults (such as address/control feedback). This is different for storage related to peripheral modules, which in general use ECC without address error protection.

3.3.3 All-X words and ECC

There is a special case for legal ECC values in the MPC5744P. Memory entries that are all zeros (All-0) or all ones (All-1), including the ECC parity bits, are not legal for memory that is checked by ECC. The flash memory allows All-1, corresponding to the status of an erase block, as a valid codeword.

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

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

Notice that for flash memory, additional dedicated safety mechanisms exist to detect failures that have the potential of leading to an All-1 word (see the "Flash Memory Controller (PFLASH)" chapter in the *MPC5744P Reference Manual* for more details on flash memory safety mechanisms).

3.3.4 ECC failure handling

Single-bit and double-bit errors (correctable and uncorrectable errors) are signaled to the FCCU unless filtered by the MEMU. The MEMU (see Memory Error Management Unit (MEMU)) may filter ECC error notification for known ECC error addresses (known permanent correctable errors). Actual implementation will signal errors, not to the FCCU, but to the MEMU which filters, then forwards unfiltered notifications in an aggregated manner to the FCCU.

3.4 Clock and power monitoring

3.4.1 Clock

Clocks in the MPC5744P are supervised by Clock Monitor Units (CMUs). The CMUs are driven by the IRCOSC (16 MHz internal oscillator) for independent operation from the monitored clocks. If a supervised clock exceeds or falls below its specified frequency range on the chip, the supervising CMU flags an error that sends a signal to the FCCU.

Clocks supervised by CMU instance are as follows:

- CMU0: IRCOSC vs XOSC clock and Motor Control clock domain
- CMU1: Checker core system clock
- CMU2: Clock for peripheral bridges PBRIDGE_0 and PBRIDGE_1
- CMU3: Clock for ADC
- CMU4: Clock for SENT module

Modules that do not have CMUs or create their own internal clocks are:

- LFAST
- FlexRay
- ENET
- PIT

The clock tree for Main Core_0 and Checker Core_0 (the two cores operating in lockstep, composing the Safety Core) consists of separate clock branches that are not shared with any other module (see the "Clocking" chapter in the *MPC5744P Reference Manual* for details).

NOTE

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

3.4.2 Power

There are two types of voltage supervisors on the MPC5744P: 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)" and "Power Control Unit (MC_PCU)" chapters in the *MPC5744P 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 DSPI is provided by the MCU, two DSPI modules (DSPIn, DSPIm) are included and connected externally through different pins. Internally, DSPIn would then be connected to PBRIDGE0 and DSPIm to PBRIDGE1, and they would be accessible via different addresses.

The arrangement of I/O peripherals onto two PBRIDGEs, as well as further CCF prevention measures, allow 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 high bandwidth communication controllers additional software measures are employed that do not require redundant communication peripherals.

The following communication controllers do not contain special safety mechanisms (above what is included in them by their protocol specifications) nor are they duplicated or spread over the PBRIDGE:

- FlexRay
- CAN
- ENET

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
- ENET
- 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_MSCR*n*[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 *MPC5744P Reference Manual*).

The FCCU intends to drive FCCU_F[0] 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_F[0] to a fault condition, the SIUL2 disables the output buffer of such pins for which SIUL2_MSCR*n*[SMC] is cleared and thus disables transmission of erroneous messages until FCCU intends to drive FCCU_F[0] to a non-fault condition. After a communication controller transmission port is disabled, it remains in the same state as long as the FCCU drives FCCU_F[0] 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, CAN, ENET, FlexRay) functionality and ensure those pins do not remain in an undriven state.

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 MPC5744P, BIST is the main means to meet the requirement on latent faults as defined by the ISO 26262 standard. Different types of BIST are implemented in the MPC5744P: LBIST for digital logic, MBIST for memories, and the MCU's built-in mechanisms for testing analog peripherals. LBIST and MBIST execution is managed by the STCU2, while the testing of analog peripherals requires software intervention to be triggered (see chapter "Self-Test Control Unit (STCU2)" in the *MPC5744P Reference Manual*).

3.7.1 BIST during boot

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

- Memory BIST for all RAMs and ROM
- Scan-based Logic BIST for digital logic, which is divided into multiple partitions that can be configured to be tested in parallel or sequentially to find the best time versus power consumption trade off.

3.7.2 Online Logical BIST (LBIST)

Though LBIST and MBIST are primarily intended to be run at startup under STCU2 control, the MCU allows software to trigger a memory self-test or LBIST of one partition during runtime. Because LBIST and MBIST are destructive, a reset is performed for the tested module (memory or partition) before operation resumes. This reset always leads to a reset of the entire device. When a runtime LBIST results in a reset of the entire device, the LBIST result is accessible to software after the reset. If a reset occurs during an LBIST, the LBIST is aborted.

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_F[0] and FCCU_F[1]. FCCU_F[0] can also serve as an error input mechanism (see Fault Collection and Control Unit (FCCU) and the FCCU configuration section in the *MPC5744P Reference Manual* for details on the fault output signals).

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

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

3.8.2 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 MPC5744P 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 MCU
- Changing the state of the failure indication pins, FCCU_F[*n*]
- Disabling the transmission capabilities of communication controllers (for example, FlexRay, CAN, 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 *MPC5744P 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 Memory Error Management Unit (MEMU)

The MEMU is responsible for the collection and reporting of error events associated with ECC logic used on SRAM (a system RAM in this context is any RAM that is CPU accessible), peripheral RAM, and flash memory. When ECC error events occur, the MEMU receives an error signal that causes an event to be recorded, and possibly sets corresponding error flags that are reported to the FCCU.

The MEMU stores the addresses of ECC errors that have occurred in a table as follows:

- Uniqueness of the addresses in the table is ensured. For example, if an address is already stored in the MEMU's table, correctable errors at that address will no longer be reported to the FCCU.
- Software can read and modify the MEMU table and remove individual entries (by marking them as invalid). For example, this allows software to invalidate a new entry in the MEMU table and wait for a repeat occurrence of the ECC error indicating a permanent error in memory.
- If the MEMU table overflows, an additional signal (instead of the ECC error signal) is sent to the FCCU.

As the MEMU aggregates address information from several sources, it might not be able to process all simultaneously arriving reports. This is called a simultaneous overflow and is signaled to the FCCU as a buffer overflow (see the "Memory Error Management Unit (MEMU)" chapter in the *MPC5744P Reference Manual*).

3.9.1 Interface to ECC units

The MEMU receives data according to the following interface per ECC unit connected:

- Whether the error is a Single-Bit-Error (SBE) or Uncorrectable Error (UCE) type
- Address of the memory where the error occurred
- A configuration specifying whether the ECC unit is attached to a safety-related system RAM, to flash memory, or to a peripheral RAM

If an error is signaled, it is compared against all errors known for that storage:

NOTE

If a previously known error has been marked invalid by software, no comparison against the address will occur.

- If no entry of that address is already in the buffer, it is to be added into the appropriate table depending on the SBE versus UCE indicator

NOTE

For this comparison the bit position will not be taken into account. This is in contrast to MBIST reporting.

- If there is no free entry left in that buffer, the overflow flag is set.

If a valid entry of that address is in the buffer, the following occurs:

- If the entry indicates an SBE in that address, and the error indicated is uncorrectable, a new entry is added in the MEMU buffer.
- In all other cases, nothing is changed in the entry.

3.10 Common cause failure measures

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

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

3.11 Operational interference protection

Being a multi-master system, MPC5744P provides safety mechanisms to prevent non-safety masters from interfering with the operation of the core, as well as mechanisms to handle the concurrent execution of software with different (lower) ASIL. Interference freedom is guaranteed via a hierarchical memory protection schema including:

- MPUs
- PBRIDGEs
- Register protection.

There are two Memory Protection Unit levels included in the MPC5744P. The Core Memory Protection Unit (CMPU) is a mechanism included in each core 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 System MPUs (SMPU) located in each XBAR. They will prevent access of different bus masters to address ranges and will typically be used by the safety application to prevent non-safety related modules access to the application's safety-relevant resources.

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

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

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 MPC5744P.

The MPC5744P 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 MPC5744P, 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 MPC5744P 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 MPC5744P 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 MPC5744P. 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).

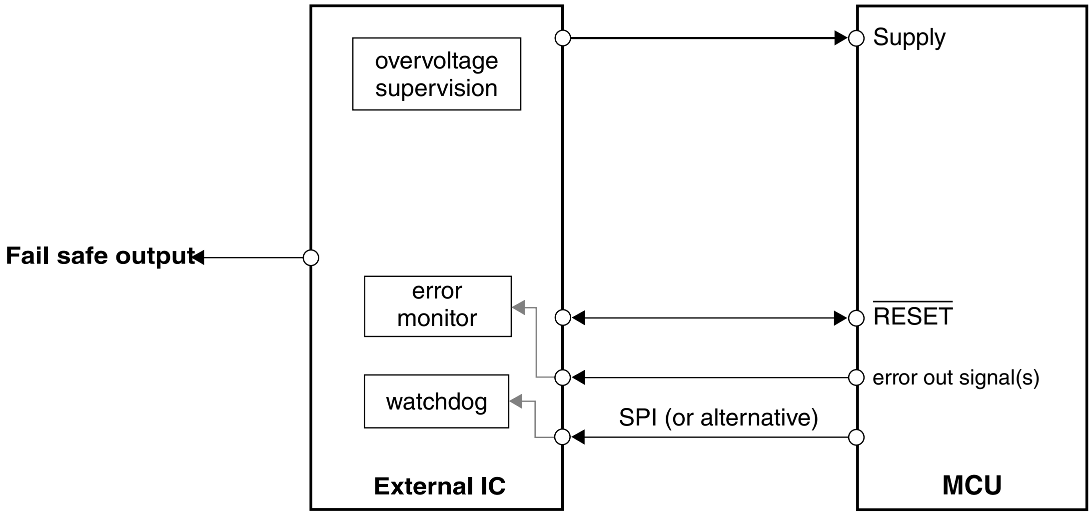


Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the MPC5744P for safety-related systems.

It should be noted that failure modes of external services are only partially considered in the FMEDA of the MPC5744P (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 MPC5744P is considered to be in a Safe state$_{MCU}$ (for example, unpowered and outputs tristated), the system containing the MPC5744P may not be compliant with the Safe state$_{system}$. A possible system level safety measure to achieve Safe state$_{system}$ may be to place pull-up or pull-down resistors on I/O when the high-impedance state is not considered safe.

Assumption: [SM_038] If a high-impedance state on an output pin is not safe, pull-up or pull-down resistors shall be added to safety-related outputs. The need for this will be application dependent for the unpowered or reset (tristated I/O) MPC5744P.[end]

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

4.1.1.2 External Watchdog (EXWD)

An external device, acting as an independent timeout functionality (for example, External Watchdog (EXWD)), should be used to cover Common Mode Failures (CMF) of the MPC5744P for safety-related systems.

The trigger may be a discrete signal(s) or message object(s). If within a defined timeout period the EXWD is not triggered, a failure will be considered to have occurred which would then switch the system to a Safe state$_{system}$ within the FTTI (for example, the EXWD disconnects the MPC5744P from the power supply, or communication messages are invalidated by disabling the physical layer driver).

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

The implementation of the communication between the MPC5744P 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 MPC5744P, even if it is held in reset.

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

The *MPC5744P 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, *MPC5744P* 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 *MPC5744P* even if kept in reset.

Implementation hint: An external and independent device may provide an over voltage monitor for the external MPC5744P supplies. If the supplied voltage supply is above the recommended operating voltage range of the MPC5744P, the MPC5744P should be maintained with no power. The external power supply monitor will switch the system to a Safe $state_{system}$ within the FTTI, and maintain it in Safe $state_{system}$ (for example, over-voltage protection with functional safety shut-off, or a switch-over to a second power supply unit).

If the MPC5744P power supply can be designed to avoid any potential of over-voltage, the external voltage monitoring can be excluded from the system design.

Over-voltage on some supplies will be detected by the MPC5744P itself, but system level measures might be required to maintain the Safe $state_{system}$ in case an over-voltage situation may cause damage to the MPC5744P.

4.1.1.4 Error Out Monitor (ERRM)

If the MPC5744P signals an internal failure on its error out signals (FCCU_F[0], and/or FCCU_F[1]), the system may no longer rely on the integrity of the other MPC5744P outputs for safety functions. If an error is indicated, the system has to switch to, and remain in, Safe $state_{system}$ without relying on the MPC5744P. 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 $state_{system}$ 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 *MPC5744P 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_F[0] (or FCCU_F[1]), is connected to a separate device.

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

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

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

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

The advantage of a single 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_F[0], or FCCU_F[1], 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_F[0], or FCCU_F[1]

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_F[0] as an input and FCCU_F[1] as an output. This test mode can be used to check the state of the FCCU_F[0] input by reading FCCU_EINOUT[EIN0]. Likewise, the user can control the FCCU_F[1] output by modifying FCCU_EINOUT[EOUT1].

Since the FCCU will be monitoring the system, it is sufficient to check FCCU_F[0] (or FCCU_F[1]) 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_F[0] (or FCCU_F[1]) to have a lower drive strength, and the error output signal is configured with highest drive strength. [end]

Using a lower drive strength on the GPIO near FCCU_F[0] (or FCCU_F[1]) 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_F[0], or FCCU_F[1]), is connected to a separate device applying time domain coding (for example, a decoder), a window timeout or windowed watchdog function, is good practice.

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

4.1.2 Optional hardware measures on system level

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

4.1.2.1 External communication

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

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

4.1.2.2 PWM output monitor

The MPC5744P timer modules may require system-level safety measures in order to achieve high functional safety integrity levels.

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

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

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

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

- Dead-time may need to always be positive, and greater than the maximum value of T_{ON} or T_{OFF} of the inverter switches.
- Open GPIO, and shorts to supply or ground, may need to be detected. This can be accomplished, for example, by an external feedback mechanism to a timer module of the MPC5744P capable of performing input capture functionality.

The system must be switched to Safe $state_{system}$ if the MPC5744P detects an error.

To reduce the likelihood of erroneous control (for example, a motor control application with dead-time requirements to reduce the likelihood of short circuits destroying the motor) in functional safety applications using I/O to control an actuator with a short FTTI, functional safety requires system level supervision if the maximum fault indication time and fault reaction time of MPC5744P exceeds the FTTI of the actuators.

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

4.2 PowerSBC

The system basis chips MC33907 and MC33908 (PowerSBC) from NXP are ideally suited to be used in combination with MPC5744P to serve as a separate device as mentioned in Assumed functions by separate circuitry.

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

Figure 4-2 depicts a simplified application schematic for a safety-related system in conjunction with the MPC5744P.

Out of a single battery supply with a wide voltage range (V_{SUP}, 3.5 V…28 V), the MC33907/08 generates 5 V (V_{CCA}) and 3.3 V (V_{CORE}) to supply the MPC5744P as well as an auxiliary voltage (V_{AUX}) to supply other devices (for example, sensors or separate

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

The MC33907/08 also monitors the state of the error out pins FCCU_F[0] and FCCU_F[1], using the bistable protocol. Via SPI, the MPC5744P repetitively triggers the windowed watchdog of the MC33907/08 with a valid answer. A dedicated fail safe state machine is implemented to bring and maintain the application in Safe $state_{system}$. In case of a failure (for example, the watchdog is not serviced correctly), RSTb is asserted low to reset the MPC5744P. A fail-safe output (FS0b) is available to control or deactivate any fail-safe circuitry (a power switch, for example). Another fail-safe output is available with PWM encoding for error indication (a warning lamp, for example). MC33907/08 also includes hardware Built-In Self-Tests (BIST).

An interrupt output (INTb) is connected to an IRQ input of the MPC5744P.

By a connection of the signal MUX_OUT to an ADC input of MPC5744P, further diagnostic measures are possible (for example, reading temperature or measuring V_{BATT}). Digital inputs (IO_4, IO_5) may be used for monitoring error signal handling of other devices. Additionally, MC33907/08 may act as a physical interface to connect the MPC5744P directly with a CAN or LIN bus.

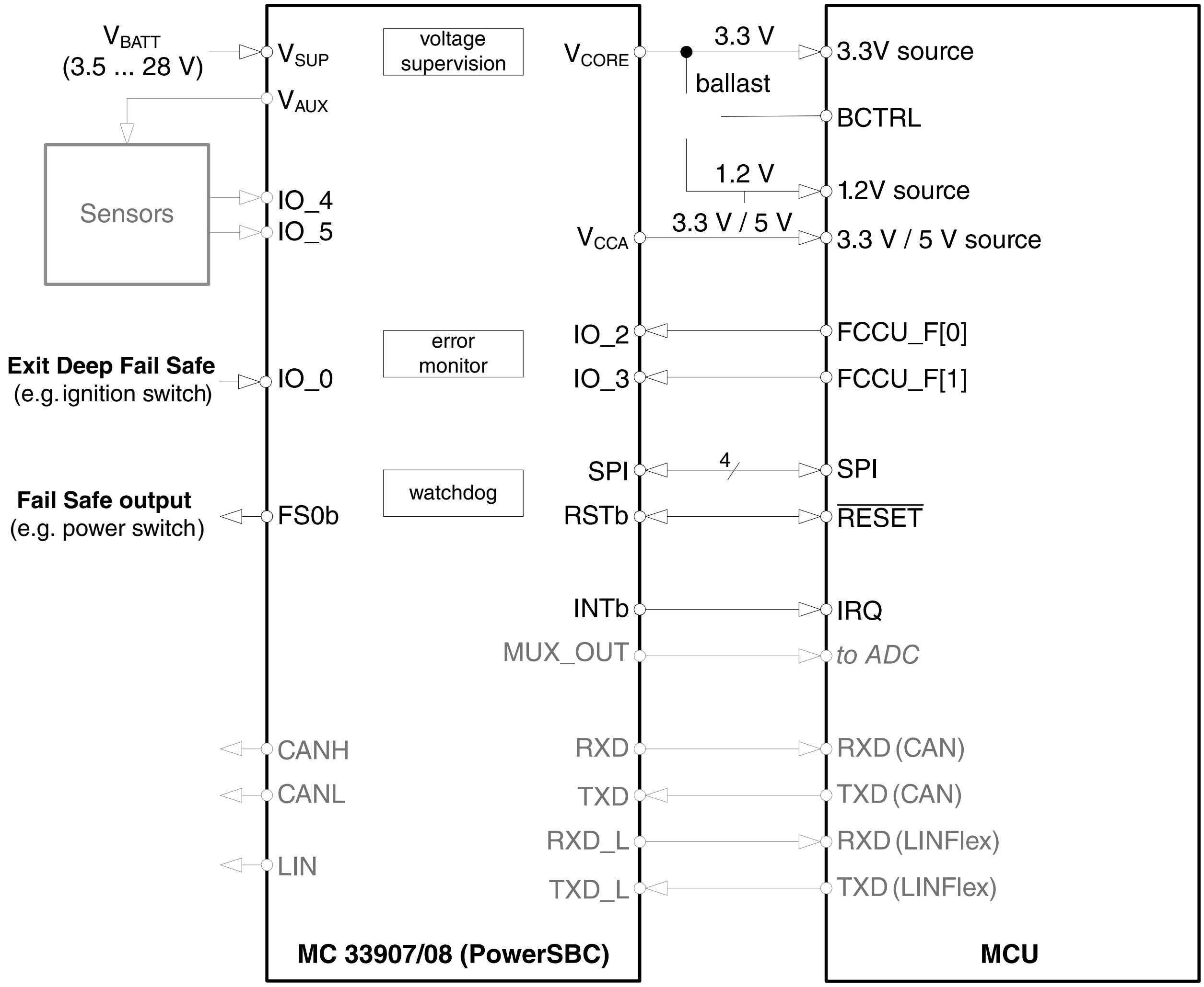


Figure 4-2. Functional safety application with PowerSBC

NOTE

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

Chapter 5
Software requirements

5.1 Software requirements on system level

This section lists required, or recommended, safety measures which should be in place when using the MPC5744P in safety systems.

The MPC5744P on-chip modules not explicitly mentioned here do not require specific safety measures to be used in safety systems. The modules that are replicated 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 the internal regulator for the logic power supply (1.25 V) and a set of voltage monitors. Particularly, it embeds low voltage detectors (LVD) and a high voltage detector (HVD). If one of the monitored voltages goes below (LVD) or above (HVD) given thresholds, a destructive reset is initiated to control erroneous voltages before these could cause a potential failure (for correct operating voltage ranges please see the *MPC5744P Data Sheet*).

To ensure functional safety, the PMC monitors various supply voltages of the MPC5744P device (as seen in Table 5-1).

Assumption: [SM_144] The application software must initiate the hardware-assisted self-test to detect LVD/HVD failures after startup. [end]

Assumption: [SM_084] The application software must check the status registers of the FCCU and MC_RGM for the results of the hardware-assisted self-test. [end]

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

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

The PMC BISTs are automatically run during startup, but the LVDs and HVDs are disabled until after testing completes.

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 MPC5744P (the FCCU). The LVDs and HVDs also report errors to the FCCU, but since the LVD and HVD errors are handled by the MC_RGM, the FCCU error reporting is not utilized.

Note

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

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

Table 5-1. PMC monitored supplies

Detector Name	Voltage Monitored
LVD_FLASH	3.3 V Flash supply
LVD_IO	3.3 V I/O supply
LVD_PMC	3.3 V VREG supply
LVD_CORE	1.25 V core supply
HVD_CORE	1.25 V core supply
LVD_ADC	3.3 V ADC supply
LVD_OSC	3.3 V OSC supply

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

5.2.1.1 1.25 V supply supervision

Voltage detectors LVD_CORE and HVD_CORE monitor the digital (1.25 V) core supply voltage for over and under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case the core main voltage detector detects over or under voltage during normal operation of the MPC5744P, a destructive reset, a functional reset, or an interrupt is triggered.

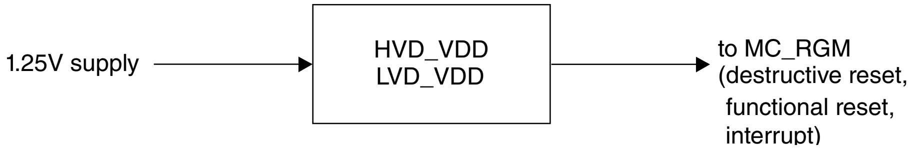


Figure 5-1. Logic scheme of the core voltage detectors

By this means, a failing external ballast transistor (stuck-open, stuck-closed) is also detected.

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

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

Implementation hint: The digital (1.25 V) core supply voltage may be monitored externally and the power supply shut down in case of an over voltage.

5.2.1.2 3.3 V supply supervision

Voltage detectors LVD_FLASH, LVD_IO, LVD_PMC, LVD_ADC and LVD_OSC monitor the 3.3 V supplies for under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case a single LVD detects under voltage during normal operation of the MPC5744P, a destructive reset is triggered.

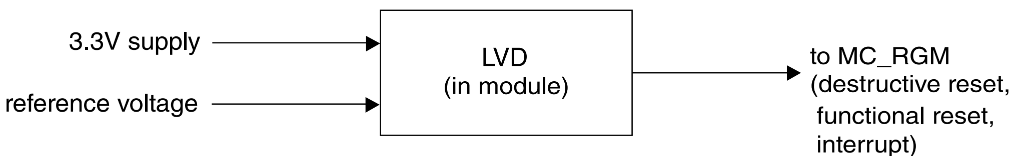


Figure 5-2. Logic scheme of the 3.3 V voltage detectors

5.3 Clock

5.3.1 Dual PLL Digital Interface (PLLDIG)

The MPC5744P consists of two PLLs used to generate high speed clocks, an FMPLL (PLL1) (which provides a frequency modulated clock) and non-FMPLL (PLL0). The FMPLL and non-FMPLL provide a loss of lock error indication that is routed to the MC_RGM and the FCCU (NCF[25], NCF[26]). If there is no PLL lock, the system clock can be driven by the IRCOSC. Glitches which may appear on the crystal clock are filtered (low-pass filter) by the FMPLL. The FMPLL dedicated to the system clock is a frequency modulated PLL to reduce EMI, and is distributed to most of the MPC5744P modules. The auxiliary clock from the non-FMPLL is not modulated, and is distributed to those peripherals that require precise timing.

5.3.1.1 Initial checks and configurations

After system reset, the external crystal oscillator is powered down and the PLLs are deactivated. Software shall enable the oscillator. After system reset, the MPC5744P uses the internal RC oscillator clock (IRCOSC) as its clock source (see the "Clocking" and "IRCOSC Digital Interface" chapters in the *MPC5744P Reference Manual* and Internal RC Oscillator for details on IRCOSC configuration).

Assumption: [SM_078] Before executing any safety function, a high quality clock (low noise, low likelihood for glitches) based on an external clock source shall be configured as the system clock of the MPC5744P. [end]

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

Implementation hint: The two PLLs can be configured to use the external oscillator (XOSC) as a clock reference, or an internally provided clock reference. In general MC_CGM_AC3_SC[SELCTL] and MC_CGM_AC4_SC[SELCTL] shall be set to 1.

Implementation hint: PLLDIG_PLL0SR[LOLF] and PLLDIG_PLL1SR[LOLF] indicates that a loss of lock event occurred. The PLLDIG_PLL0CR[LOLIE] and PLLDIG_PLL1CR[LOLIE] can be set to enable an interrupt request upon loss of lock.

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

Rationale: To reduce the impact of glitches stemming from the external crystal and its hardware connection to the MPC5744P.

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

Implementation hint: Either during or after initialization, but before executing any safety function, application software can check the current system clock by checking the MC_ME_GS[S_SYSCLK] flag. MC_ME_GS[S_SYSCLK] = 4 indicates that the FMPLL clock is being used as the system clock.

5.3.2 Clock Monitor Unit (CMU)

At startup, the CMUs are not initialized and the IRCOSC is the default system clock. Stuck-at faults on the external oscillator (XOSC) are not detected by the CMUs at power-on since the monitoring units are not initialized and the MPC5744P is still running on the IRCOSC.

The CMUs are driven by the 16 MHz internal reference clock oscillator (IRCOSC) to ensure independence from the monitored clocks. CMUs flag errors associated with conditions due to clock out of a programmable bounds and loss of reference clock. If a supervised clock leaves the specified range for the device, an error signal is sent to the FCCU. MPC5744P includes the CMUs shown in Table 5-2.

Table 5-2. Clock Monitoring Units

CMU	Monitored Clock
CMU_0	Loss of XOSC MOTC_CLK frequency out of range
CMU_1	Core frequency out of range
CMU_2	PBRIDGE frequency out of range
CMU_3	ADC frequency out of range
CMU_4	SENT frequency out of range

The CMUs use the IRCOSC (16 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 (XOSC)

- loss of reference (IRCOSC)
- PLL clock out of a programmable frequency range (frequency too high or too low)
- loss of PLL clock

The 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 Non-Critical Faults Mapping" table shown in the *MPC5744P Reference Manual*).

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

5.3.2.1 Initial checks and configurations

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

Rationale: To monitor the integrity of the clock signals

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

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

- The first configuration is related to the crystal oscillator clock (XOSC) monitor of CMU_0. Software configures CMU_0_CSR[RCDIV] to select an IRCOSC divider. The divided IRCOSC frequency is compared with the XOSC.
- The second configuration is related to other clock signals being monitored. The high frequency (CMU_*n*_HFREFR_A[HFREF_A]) and low frequency references (CMU_*n*_LFREFR_A[LFREF_A]) are configured.

Once the CMUs are configured, clock monitoring will be enabled when software writes CMU_*n*_CSR[CME_A] = 1.

5.3.3 External Oscillator (XOSC)

FlexRay and CAN each feature modes in which they are directly clocked from the XOSC.

5.3.3.1 Initial checks and configurations

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

5.3.3.2 Runtime checks

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

5.3.4 Internal RC Oscillator

The Internal RC Oscillator (IRCOSC) has a nominal frequency of 16 MHz, but frequency accuracy over the full voltage and temperature range has to be taken into account (see the *MPC5744P Data Sheet*). Functional safety-related modules which use the clock generated by the IRCOSC are:

- FCCU
- CMU
- SWT

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

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 IRCOSC. This feature allows measurement of the IRCOSC frequency using the XOSC as the reference (IRC_SW_CHECK).

Assumption: [SM_073] The IRCOSC frequency is measured and compared to the expected frequency of 16 MHz. This test is performed after power-on, but before executing any safety function. Software writes CMU_CSR[SFM] = 1 to start the frequency measurement, and the status of the measurement is checked by reading this same field. When as CMU_CSR[SFM] = 0 the frequency measurement has completed (see "Frequency meter" section in the "Clock Monitor Unit (CMU)" chapter of the *MPC5744P Reference Manual* for details.). [end]

Rationale: To check the integrity of the IRCOSC

Note

If the IRCOSC is not operating due to a fault, the measurement of the IRCOSC 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 IRCOSC. This feature allows measurement of the IRCOSC frequency using the XOSC as the clock source.

Assumption: [SM_074] To detect failure of the IRCOSC, the application software shall utilize frequency metering of CMU_0 to read the IRCOSC frequency and compare it against the expected value of 16 MHz[1]. [end]

Implementation hint: See the **Assumption:** in Initial checks and configurations for an explanation on how to use CMU_0 to check the IRCOSC.

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

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

1. Nominal frequency of the IRCOSC is 16 MHz, but a post trim accuracy over voltage and temperature must be taken into account (see the *MPC5744P Data Sheet*).

5.4 Flash

5.4.1 Flash memory

The MPC5744P provides programmable non-volatile flash memory (NVM) with ECC, which can be used for instruction and/or data storage.

The correct operation of ECC logic is guaranteed by EDC after ECC and latent faults are detected by the execution of the LBIST.

5.4.1.1 EEPROM

The MPC5744P provides four blocks (2 x 16 KB and 2 x 32 KB) of the flash memory for EEPROM emulation. ECC events detected on accesses to the EEPROM flash memory blocks are not reported to the Memory Management Unit (MEMU). Single-bit errors are corrected but not signaled to the MEMU. Multi-bit errors are replaced by a fixed word (representing an illegal instruction) and are also not forwarded to the MEMU.

Assumption: [SM_114] The software using the EEPROM for storage of information will use checks to detect incorrect data returned from the EEPROM emulation. [end]

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

5.4.1.2 Initial checks and configurations

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

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

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

Implementation hint: This test may be started by application software: its result may be validated by reading the corresponding registers in the flash memory controller after it has been finished (see "Array integrity self check" section in the "Flash memory" chapter of the *MPC5744P Reference Manual*).

5.4.1.3 Runtime checks

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

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

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

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

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

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

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

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

Implementation hint: The flash memory ECC fault report check is executed in software . The test consists of software reading from the flash memory UTest area (see "UTEST flash memory map" table in the "Memory map" chapter of the *MPC5744P Reference Manual*). a set of test patterns to test the integrity of the ECC logic fault reporting path to the MEMU and FCCU (executed at start-up, latent failure measure).

5.5 SRAM

5.5.1 End-to-end ECC (e2eECC)

The MPC5744P includes end-to-end ECC (e2eECC) support for improved functional and transient fault detection capabilities. Memory-protected by the traditional ECC/EDC generates and checks additional error parity information local to the memory unit to detect and/or correct errors which have occurred on stored data in the memory.

In contrast, in the MPC5744P e2eECC protected memory, the bus master initiates the data write and generates ECC checkbits based on 29-bit address and 64-bit data fields. The data including the checkbits are transferred from the bus master to the appropriate bus slave. Both data and checkbits are stored into the memory. When the bus master initiates a read of previously written memory location, the read data and checkbits are passed onto the system bus interconnection. The bus master captures the read data and associated checkbits, performs the ECC checkbit decode and syndrome generation and performs any needed single-bit correction.

The e2eECC provides:

- ECC for master-slave accesses via the crossbar
- ECC is stored in the memories on write operations and validated by the crossbar master on every read operation
- ECC bits are stored alongside data in Flash memory and RAM.
- ECC on address and data covers 64-bit data and 29-bit address bits

All-X errors in memory have special handling as it is thought that there may be a higher probability of All-X errors than random wrong bits.

The ECC used for flash memory marks All-0 as being in error, but allows All-1 situations to take into consideration reading erased, uninitialized flash memory. Flash memory includes additional mechanisms to detect faults that could lead to array wide All-1 situations.

The ECC for RAM, without inclusion of address, mark All-X as errors.

The ECC for RAM, with inclusion of address, cannot guarantee that All-X is an error for any address because All-0 and All-1 will be correct codewords for approximately every 256th address. In these RAMs, at more than every 2nd address, All-1 and All-0 will be uncorrectable errors. It is possible to read such an address where All-X is uncorrectable periodically to determine situations in which an error causes a whole RAM block to become All-X. Testing All-X in RAM defines an algorithm to determine such addresses.

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 Debug mode

The debugging facilities of the MPC5744P 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 MPC5744P 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. 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.

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 any modules that can be frozen in debug mode, such as the following:

- Software Watchdog Timer (SWT)
- System Timer Module (STM)

- Deserial Serial Peripheral Interface (DSPI)
- Periodic Interrupt Timer (PIT)
- Fault Collection and Control Unit (FCCU)
- FlexRay
- CAN

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

CAN_ MCR[FRZ] controls CAN Module behavior in the debug mode. If the CAN_ MCR[FRZ] = 0, the CAN 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.

The Interrupt Controller (INTC) operation in debug mode is identical to its operation in normal mode. No specific action is required by application software.

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

When ETIMER_CH*n*_CTRL3[DBGEN] = 00, the eTimer continues normal operation while the device is in debug mode.

When FlexPWM_SUB*n*_CTRL2[DBGEN] = 1, the Motor Control Pulse Width Modulator Module (FlexPWM) continues to run 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.

SRX_GBL_CTRL[DBG_FRZ] controls the SENT behavior during debug mode. If the SRX_GBL_CTRL[DBG_FRZ] = 0 (cleared), the SENT will not freeze during debug mode.

5.6.1.2 Test mode

Several mechanisms of the MPC5744P 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 VPP_TEST pin is for test purposes only, and must be tied to GND during normal operating mode. From a system level point of view, measures must ensure that the VPP_TEST pin is not connected to V_{DD} during boot to avoid entering test mode. The activation of test mode is supervised by the FCCU and will signal a fault condition when test mode is entered.

5.6.2 Additional configuration information

5.6.2.1 Stack

Stack overflow and stack underflow is a common mode fault due to systematic faults within application software. A stack overflow occurs when using too much memory (pushing to much data) on the stack. A stack underflow occurs when reading (pop) too much data from memory. The stack contains a limited amount of memory, often determined during development of the application software. When a program attempts to use more space than is reserved (available) on the stack (when accessing memory beyond the stack's upper and lower bounds), the stack is said to overflow or underflow, typically resulting in a program crash.

It may be beneficial to implement a measure supervising the stack and respectively generating a fault signal in case of stack overflow and stack underflow.

5.6.2.1.1 Initial checks and configurations

Assumption under certain conditions:[SM_139] When stack underflow and stack overflow due to systematic faults within the application software endangers the item (system) level, functional safety mechanisms may be implemented to detect stack underflow and stack overflow faults. [end]

Rationale: To have a notification in case of stack overflow or stack underflow error

Implementation hint: Data Address Compare 1 (DAC1) and Data Address Compare 2 (DAC2) Special Purpose Registers (SPRs) may be used for incremental stack overflow or stack underflow detection when not being used as a hardware or software debug resource. Stack limit checking is available regardless of External Debug Mode (EDM) or Internal Debug Mode (IDM), and when resources used for stack limit checking are software controlled, will utilize a Data Storage Interrupt (DSI) or machine check exception.

A data address compare (DAC) exception is signaled when there is a data access address match as defined by the debug control registers and data address compare events are enabled. This could either be a direct data address match or a selected set of data addresses, or a combination of data address and data value matching. The debug interrupt is taken when no higher priority exception is pending.

Software-owned stack limit checking does not require IDM to be set. Hardware owned stack limit checking requires EDM to be set. When stack limit checking is enabled, and DAC resources used for stack limit checking are owned by software, DAC events are not generated for resources configured to perform stack limit checking, and no DBSR DAC status flag will be set due to a detected stack limit violation.

Instead, depending on the processor mode, a data storage interrupt or a machine check exception is signaled. When stack limit checking is enabled, and DAC resources used for stack limit checking are owned by hardware, DAC events will be generated for resources configured to perform stack limit checking, and the EDBSR0 DAC status flag will be set due to a detected stack limit violation, causing entry into debug halted mode in the same way as a DAC exception normally does. The only difference is that qualification of the access address is performed as discussed in the next paragraph.

Incremental stack limit checking may be implemented using two data address watchpoints defined by DAC1 and DAC2. As hardware does not qualify a load or store access address with the use of GPR R1 as the base or index register used to compute an effective address when a load or store instruction is executed, special care has to be taken the watchpoints are not used elsewhere in the application software (guard band address range). This measure does only enable incremental stack overflow, as it only detects data addressing of the limit (upper and lower) address. Addressing going beyond the limits will be undetected. When DAC resources configured to perform incremental stack limit checking are not owned by hardware, if a stack limit violation occurs when performing the load or store, the access is aborted, and an error report machine check is generated, with MCSRR0 pointing to the address of the load or store access which generated the stack overflow/underflow. If DAC resources configured to perform stack limit checking are owned by hardware, then a normal DAC event is generated (but qualified with use of GPR R1), and debug mode entry will occur in the same manner as for a non-stack limit DAC event.

When stack limit checking is enabled for a stack access, and DAC*n* resources are owned by hardware, the EDBSR0 DAC status flag will be set due to a detected stack limit violation, to cause entry into debug halted mode or to generate a watchpoint, or both, i.e. after the access has completed.

Independent limit checks for supervisor and user accesses may be implemented by allocating independent DACn resources to each, or a single limit may be applied using a single DACn resource. If more than one DACn resource is utilized, a DAC hit on any resource utilized for stack limit checking will cause the corresponding stack limit exception action to occur. If both a hardware-owned and a software-owned resource generate a stack limit exception for a given load or store, the software resource will have priority, since it is detected prior to completion of the access, and the access is aborted, thus the hardware event will not occur.

Note

For DAC1 and DAC2, access type (read, write) control is part of DBCR0.

5.6.2.2 MPC5744P configuration

Assumption:[SM_140] It is required that application software verifies that the initialization of the MPC5744P 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.

Below is a list of the minimum number of checks by safety integrity functions which need to pass before executing any safety function:

- Lock-step mode check
- STCU check
- Flash Array Integrity Self check
- SUPPLY SELF-TEST
- Temperature sensor check
- SWT enabled
- CMU check
- IRC_SW_CHECK

- PMC check
- FCCU_F[*n*] signal check[2]

Prerequisites are not listed. If any of these checks fails, functional safety cannot be ensured.

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

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

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

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

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

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

Rationale: To reduce the risk accidental writes to configuration registers affecting the execution of the MPC5744P's safety function or disable the safety mechanism due to their change.

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

Rationale: To reduce the risk accidental writes configuration registers affecting the execution of the MPC5744P's safety function or disable the safety mechanism due to their change.

Implementation hint: Each of the peripheral registers that may be protected through the REG_PROT has a Set Soft Lock bit in the Register Protection space. This bit may be asserted to enable the protection of the related peripheral registers.

The Hard Lock bit (REG_PROT_GCR[HLB] = 1) may be set for best write protection.

2. Required for single FCCU signal usage only

5.6.3 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 cores and the master and checker eDMA units, respectively. The error information is forwarded to the FCCU. The RCCUs are automatically enabled when MPC5744P is in LSM mode.

NOTE

On this chip disabling lockstep mode does not free the checker core for independent execution (called DPM on other chips).

5.6.3.1 Initial checks and configurations

The use of the RCCU is indispensable, and is automatically managed by the MPC5744P. The RCCU cannot be disabled by application software during runtime. Consequently, the respective FCCU input should not be disabled.

However, LSM can be disabled during boot by reprogramming the flash memory. LSM is also disabled when Core_0 enters debug mode. If disabled, the Checker Core (Core_0 - Checker) and the RCCUs are constantly off. No dynamic switching is possible between lockstep on/off, a reset is needed to reestablish LSM. No Decoupled Parallel Mode (DPM) is available (Main Core_0 and Checker Core_0 cannot operate as two independent cores running different software). Main Core_0 is able to operate as an independent core if Checker Core_0 is disabled. The status of LSM for the Safety core can be verified by checking the Core Status register (ME_CS) in the Mode Enable module (MC_ME). If both Main Core_0 and Checker Core_0 are on, or both off, the Safety Core is in LSM.

Assumption: [SM_033] Before starting safety-relevant operations, the application software must check that the checker core is enabled and configure the FCCU to react to LSM being disabled. [end]

5.6.4 Crossbar Switch (XBAR)

The multi-port XBAR 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 XBAR 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 XBAR which are Not Safety Related shall have a lower arbitration priority on the XBAR compared to Safety Related masters. [end]

5.6.4.1 Runtime checks

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

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

The application software shall detect XBAR configuration failures once per FTTI.

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

5.6.5 Memory protection units

As a multimaster, concurrent bus system, the MPC5744P provides safety mechanisms to prevent non-safety masters from interfering with the operation of the safety core. MPC5744P also contains mechanisms to handle the concurrent operation of software tasks with different or lower ASIL classifications.

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

5.6.5.1 Core Memory Protection Unit (CMPU)

The CMPU is a MPU directly attached to each core. It is included to ensures inter-task interference protection by providing the capability of protecting regions of memory from access by software tasks with different privilege levels. The CMPU features a 24-entry region descriptor table that defines memory regions and their associated access rights. Only accesses with the sufficient rights are allowed to complete.

Using user-defined region descriptors that define memory spaces and their associated access rights, the CMPU concurrently monitors Core initiated memory accesses and evaluates the access rights of each transfer.

Assumption: [SM_092] The application shall use the CMPU to protect all memory regions that require protection against accesses from other applications. [end]

Recommendation: The CMPU should be used to ensure that only authorized software tasks can configure modules and can access only their allocated resources according to their access rights.

5.6.5.2 System Memory Protection Unit (SMPU)

The System Memory Protection Unit (SMPU) provides memory protection at the crossbar (XBAR). The SMPU allows splitting of the physical memory into 16 different regions. Each XBAR master (Core, DMA, FlexRay, Ethernet, SIPI) can be assigned different access rights to each region. The SMPU can be used to prevent non-safety masters (including DMA or FlexRay controller) from accessing restricted memory regions.

Memory accesses that have sufficient access control rights are allowed to complete, while accesses that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response. The SMPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes represent the second dimension.

Assumption: [SM_094] The SMPU shall only be programmed by the safety core. This software shall prevent write accesses to the SMPU's registers from all other masters. The SMPU programming model shall only be accessible to the safety core. [end]

5.6.5.3 Initial checks and configurations

Assumption under certain conditions: [SM_095] If non-replicated bus masters are used, system level functional safety integrity measures must cover bus operations to reduce the likelihood of replicated resources being erroneously modified. [end]

Rationale: Access restriction at the MPU level is protection against unwanted read/write accesses to some predefined memory mapped address locations by specific software routines (processes).

Implementation hint: The MPUs shall be used to ensure that only authorized software routines can configure modules and all other bus masters (SIPI, eDMA, core, FlexRay protocol controller) can access only their allocated resources according to their access rights. For the non-replicated master FlexRay, a correct MPU setup is highly recommended.

5.6.6 Interrupt Controller (INTC)

The Interrupt Controller (INTC) provides the ability to prioritize, block, and direct Interrupt Requests (IRQs). The INTC 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. The Interrupt Controller (INTC) can drop, delay or create spurious interrupts.

Assumption: [SM_098] Application software will detect the critical failure modes of the INTC for all interrupts not supervised by the high priority interrupt monitor. [end]

Implementation hint: One way to detect spurious or multiple unexpected interrupts is for the application software to read the interrupt status register of the corresponding peripheral before executing the Interrupt Service Routine (ISR). This checks that the respective peripheral has really requested an interrupt.

5.6.6.1 Periodic low latency IRQs

The Interrupt Control Monitor (INTCM) 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]

5.6.6.2 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 INTC clock cycles for the monitored interrupt.

A supervisor module configured to react to any one of the IRQ signals checks that the INTC 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.

5.6.6.3 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.

5.6.7 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 module in lockstep, 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.

5.6.7.1 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 XBAR, 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.
- If the eDMA is used to manage the analog acquisition with the CTU and ADC, the number of the converted ADC channels is saved into the CTU FIFO together with the acquired value. The eDMA transfers this value from the CTU FIFO to a respective SRAM location. Spurious or missing transfer requests can be detected by comparing the converted channel with the expected one.

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

Rationale: To reduce the likelihood of a faulty PIT (which is not redundant) from triggering an unexpected eDMA transfer

5.6.7.1.1 Peripheral lake eDMA transfers

The eDMA module is replicated but the eDMA Channel Mux, which maps the handshake signals of different peripherals to the eDMA, is not replicated. Each half of the eDMA 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 eDMA Channel Mux 1 is faulty and thus disturbs access to a peripheral in its lake, then an access triggered by eDMA Channel Mux 0 to a peripheral in its own lake will not be faulty and thus show a deviation from the faulty transfer.

5.6.7.1.2 Non-replicated eDMA transfers

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

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

5.6.8 Reset Generation Module (MC_RGM)

5.6.8.1 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) and the alternative reaction chosen in the Functional Event Alternative Request register (MC_RGM_FEAR).

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 *MPC5744P Reference Manual* for details).

5.6.8.1.1 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 MPC5744P supports functional reset escalation which can be used to generate a destructive reset if the number of functional resets reaches the programmed value. Once the functional reset escalation is enabled, the Reset Generation Module (MC_RGM) increments a counter for each functional reset that occurs between writes to the MC_RGM_FRET register. When the number of functional resets reaches the programmed value in the MC_RGM_FRET, the MC_RGM initiates a destructive reset. The counter can be cleared by software, destructive reset or power-on reset.

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

Assumption: [SM_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]

5.6.9 System timer module (STM)

5.6.9.1 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.

5.6.10 Software watchdog timer

The objective of the Software Watchdog Timer (SWT) is to detect a defective program sequence when individual elements of a program are processed in the wrong sequence, or in an excessive period of time. Once the SWT is enabled, it requires periodic and timely execution of the watchdog servicing procedure. The service procedure must be performed within the configured time window, before the service timeout expires. When a timeout occurs, a trigger to the FCCU can be generated immediately, or the SWT can first generate an interrupt and load the down-counter with the timeout period. If the service sequence is not written before the second consecutive timeout, the SWT drives its FCCU channel to trigger a fault (see "FCCU Non-Critical Faults Mapping" table shown in the *MPC5744P Reference Manual*).

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

Assumption: [SM_202] The SWT time window settings must be set to a value less than the FTTI. Detection latency shall be smaller 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 MPC5744P provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudorandom key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

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

Two service procedures are available:

- A 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 IRCOSC clock.

5.6.10.1 Run-time 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.

5.6.11 Periodic Interrupt Timer (PIT)

5.6.11.1 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

5.6.12 Mode Entry (MC_ME)

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

Implementation hint: The SWT may provide the time monitoring.

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

5.6.13 System Status and Configuration Module SSCM

5.6.13.1 Initial checks and configurations

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

Implementation hint: Execution of BAM code may be inhibited by writing SSCM_ERROR[RAE] = 1. Each access to the BAM memory area then produces an exception. This prevents accidential execution of the BAM code.

NOTE

The BAM will not execute on its own during a 'normal' boot of the MCU, but only if a serial boot, a JLR, or a test pattern load is requested.

5.6.14 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 MPC5744P.

5.6.14.1 Runtime checks

Parts of the MPC5744P 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. [end]

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

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

Implementation hint: To verify the content of the MPC5744P 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.

5.6.14.1.1 Implementation details

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

Note

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

5.6.14.1.1.1 <*module*>_SWTEST_REGCRC

The following safety integrity functions for register configuration checks are used in this document:

- ETIMER*n*_SWTEST_REGCRC

  The eTimer configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- SIUL_SWTEST_REGCRC

  The configuration registers of the SIUL2 are read and a CRC checksum is computed. The checksum is compared with the expected value.

- FLEXPWM*n*_SWTEST_REGCRC

  The FlexPWM configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- ADC*n*_SWTEST_REGCRC

The ADC configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- CTU*n*_SWTEST_REGCRC

  The CTU configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

5.6.15 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
    - 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 (10000h IRCOSC cycles) that the FCCU can be held in the configuration state.

The table "FCCU Non-Critical Faults Mapping" in chapter "Chip Configuration" of the *MPC5744P 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$_{system}$ condition within the FTTI (for example, the separate device disconnects the MPC5744P device or an actuator from the power supply).

5.6.15.1 Initial checks and configurations

Besides the possible initial configuration, no intervention from the MPC5744P 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$_{system}$ 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. This chip's reference manual includes either a table or an attached file that documents the chip's FCCU fault mapping (the source module and other information about each FCCU fault input) so you can determine the appropriate FCCU configuration for your specific application.

If the MPC5744P signals an internal failure via its error out signals (FCCU_F[*n*]), the system can no longer safely use the MPC5744P safety function outputs. If an error is indicated, the system has to be able to remain in Safe state$_{system}$ without any additional action from the MPC5744P. Depending on its functionality, the system might disable or reset the MPC5744P as a reaction to the indicated error.

5.6.15.2 Runtime checks

If the MPC5744P is continuously switching between a standard operating state and reset, or fault state, without a device shutdown, system level measures should be implemented to ensure that the system meets the Safe state$_{system}$ criteria.

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

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

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

NOTE

Longer reset cycles 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 Memory Error Management Unit (MEMU)

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

5.6.17 Error reporting path tests

It is possible to use fake fault injection to check the correct operation of several reporting paths from supervisors to the MEMU. The FCCU input table specifically lists those inputs in the 'Suggested fault reaction' column in the "FCCU Non-Critical Faults Mapping" table shown in the *MPC5744P Reference Manual*.

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

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 Flexray/CAN SRAM and System SRAM/ local RAMs/Caches to check the reporting of such errors through the MEMU to the FCCU.

A multiple cell failure caused for example, by a neutron or alpha particle or a short circuit between cells may cause three or more bits to be corrupted in an ECC-protected word. As 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 MPC5744P by the use of bit scrambling (column multiplexing) which effects, 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 3-1 for the column multiplexing factor of each memory).

5.6.18 Self Test Control Unit (STCU2)

The STCU2 executes built-in self-test (LBIST, MBIST) and gives reaction to detected faults by signaling faults to either the MC_RGM or to the FCCU (see "Self-Test Control Unit (STCU2)" in the *MPC5744P Reference Manual* for details).

5.6.18.1 Initial checks and configurations

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

Assumption under certain conditions: [SM_062] When built in self test (for example, LBIST, MBIST, ABIST) circuits of the MPC5744P are used as functional safety integrity measure (for example, to detect random faults, latent fault detection, and single-point fault detection) in a functional safety system, functional safety integrity measures on system level shall be implemented ensuring STCU2 integrity during/after STCU2 initialization but before executing a safety function. [end]

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

Implementation hint: The integrity software shall confirm that all MBISTs and LBISTs finished successfully with no additional errors flagged.

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

This is an additional functional safety layer since the STCU2 propagates the LBIST/MBIST and internal faults to the MC_RGM or the FCCU. So, reading STCU_LBS, STCU_LBE, STCU_MBSL, STCU_MBEL and STCU_ERR_STAT registers helps increasing the STCU2 self-test coverage.

Implementation hint: The STCU2 shall be configured (in test flash memory) to execute the LBIST and MBIST before activating the application safety function (see section "STCU2 Configuration Register (STCU2_CFG)" in the "Self-Test Control Unit (STCU2)" chapter of the *MPC5744P Reference Manual*).

5.6.19 Built-in Hardware Self-Tests (BIST)

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

To ensure absence of latent faults, the self-test executes both Logic Built-In Self Test (LBIST) and Memory Built-In Self Test (MBIST) during boot while the device is still under reset (offline). The boot time BIST includes the scan-based LBIST to test the digital logic and the MBIST to test all RAMs and ROMs.[3]

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

3. This does not include flash memory.

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

Assumption: [SM_109] Software shall check after MBIST execution whether two reported single-bit errors belong to the same address and thus constitute a multi-bit error. MBIST does not guarantee detection of all multi-bit errors on its own. [end]

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:

- STCU_LBS to determine which offline LBISTs failed
- STCU_LBE to determine which offline LBISTs did not finish
- STCU_MBSL, STCU_MBSM and STCU_MBSH to determine which offline MBISTs failed
- STCU_MBEL, STCU_MBEM and STCU_MBEH to determine which offline MBISTs did not finish
- STCU_LBSSW to determine which online LBISTs failed
- STCU_LBESW to determine which online LBISTs did not finish
- STCU_MBSLSW, STCU_MBSMSW and STCU_MBSHSW to determine which online MBISTs failed
- STCU_MBELSW, STCU_MBEMSW and STCU_MBEHSW – To determine which online MBISTs did not finish
- STCU_ERR_STAT – To check for internal STCU failure

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

If faults are not detected over a long time (latent faults), they can pile up once they propagate. Typically hardware assisted BIST is therefore used as safety integrity measure to detect latent faults.

The MPC5744P is equipped with a Built-in hardware self-test:

- System SRAM (MBIST, typically executed at boot-time)

- Logic (LBIST, typically executed at boot-time)
- ADC (PBIST, typically executed at start-up)
- Flash (memory array integrity self check, typically executed at start-up)
- Flash (memory margin read, typically executed following a user-detected single bit ECC correction)
- PMC (self-test of LVD/HVD, typically executed at start-up)

Boot-time test (MBIST, LBIST) are performed after the occurrence of a destructive or external reset, unless they are disabled. All boot-time tests are executed before application software starts executing. If failed, the MPC5744P will remain in the Safe state$_{MCU}$.

5.6.19.1 MBIST

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

NOTE

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

5.6.19.2 LBIST

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

NOTE

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

5.6.19.3 Flash memory array integrity self check

The flash memory array integrity self check runs in flash memory user test mode and is initiated by software. When the check has completed, software verifies the result (see the "Embedded Flash Memory (c55fmc)" chapter in the Reference Manual for more details on flash memory array integrity check).

5.6.19.4 Flash memory margin read

The flash memory margin reads may be activated to increase the sensitivity of the array integrity self check. It may be enabled in flash memory user test mode and is initiated by software (see the "Embedded Flash Memory (c55fmc)" chapter in the Reference Manual for more details on flash user margin read).

5.6.19.5 Peripheral Built-In Self-Test (PBIST)

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

5.6.19.6 PMC LVD/HVD tests

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

5.6.20 Register Protection module (REG_PROT)

The PowerPC architecture supports two levels of privilege for program execution: user mode and supervisor mode. Only the supervisor mode allows the access to the entire CPU register set, and the execution of a subset of instructions is limited to supervisor mode only. In user-mode, access to most registers including system control registers is denied. It is intended that most parts of the software be executed in user-mode so that the MPC5744P is protected from errant register changes made by other user-mode routines. User versus supervisor mode can also be used as a decision criteria in the MPUs and the peripheral access control (PAC) of the PBRIDGES.

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

- A register cannot be written once Soft Lock Protection is set. The lock can be cleared by software or by a system reset.

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

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

Assumption:[SM_125] Configuration registers, and registers that aren't modified during application execution, should be protected from unintended software write accesses (e.g. with a Hard Lock Protection). [end]

5.6.20.1 Runtime checks

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

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

Implementation hint: Most of the off-platform peripherals have their own Register Protection module. Register Protection address space is inside the memory space reserved for the peripherals (please, refer to the " MPC5744P registers under protection" section of the MPC5744P Reference Manual). Each peripheral register that can be protected through the Register Protection module has a Set Soft Lock bit reserved in the Register Protection address space. This bit is asserted to enable the protection of the related peripheral registers. Moreover, the Hard Lock Bit (REG_PROT_GCR[HLB] = 1) should be set for best write protection.

5.7 Peripheral

5.7.1 Communications

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

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

FlexRay and CAN do not have safety mechanisms other than what is included in their protocol specifications. The application software, or operating system, needs to provide the safety measures for these modules to meet safety requirements.

5.7.1.1 Redundant communication

Parts of the integrated DSPI , LINFlex and SENT 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 (DSPI)
- LINFlexD Communication Controller
- SENT

There are no special functional safety mechanisms for DSPI , SENT and LINFlexD other than what is included via the 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.

5.7.1.2 Fault-tolerant communication protocol

Portions of the integrated FlexRay, LINFlexD and CAN communication channels do not independently 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, is required for the following:

- FlexRay Communication Controller

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

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

Typically mechanisms are:

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

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

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

An alternative approach to improve the functional safety integrity of CAN may be to use multiple instances of the CAN 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.

5.7.2 I/O functions

The integrity of functional safety-relevant periphery will mainly be ensured by application level measures (for example, connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on).

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

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

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

Implementation hint: Possible measures could be using different replicated peripherals (for example, eTimer_0, eTimer_1, or FlexPWM) to implement multiple independent and different channels.

5.7.2.1 Digital inputs

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

5.7.2.1.1 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).

- Double read operation of a digital input is implemented by two general purpose inputs (GPI) of the SIUL2 unit. A comparison (by software) between the double reads (for example, reads from both GPIOs) detects an error (please refer to Figure 5-3).
- A double read PWM input is implemented by using two modules as two channels. The functional safety integrity is achieved by double reads and a software comparison. One channel is provided by eTimer_1 and the other by eTimer_0 or eTimer_2. Read PWM input means any input read related to signal transitions (rise or fall). This may also include the time that the signal was high, low or both (see Figure 5-3).

For each signal of a double read, the SIUL2 can provide additional channels to support interrupt-based reading for each signal (see Figure 5-4).

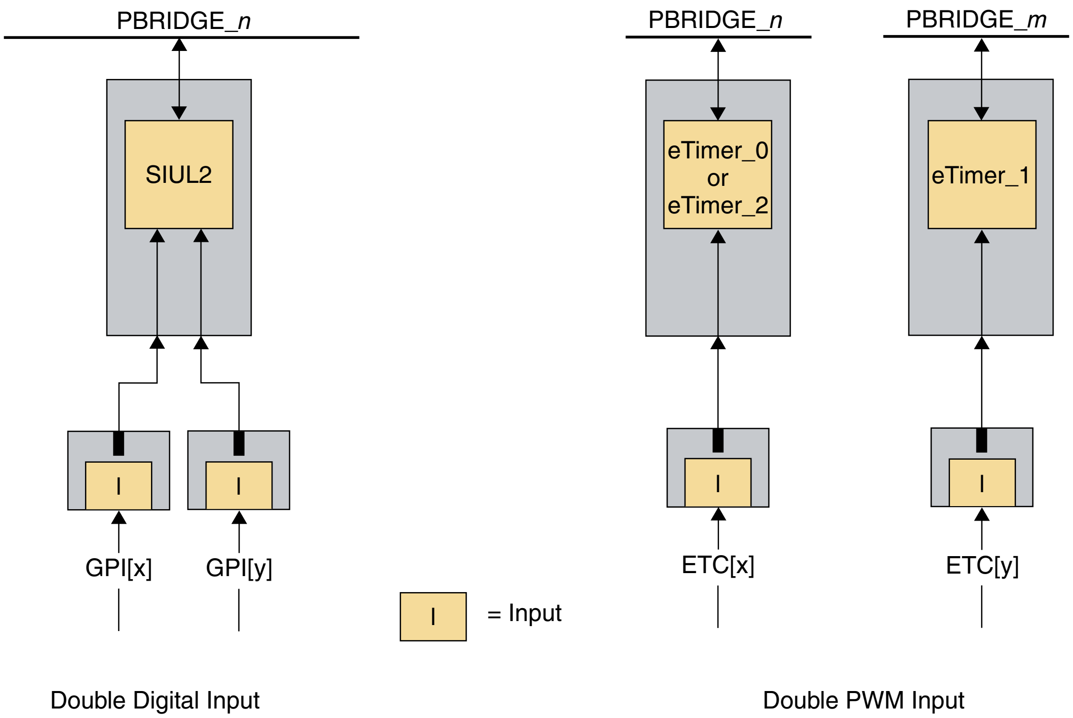


Figure 5-3. Double Digital input and Double PWM input

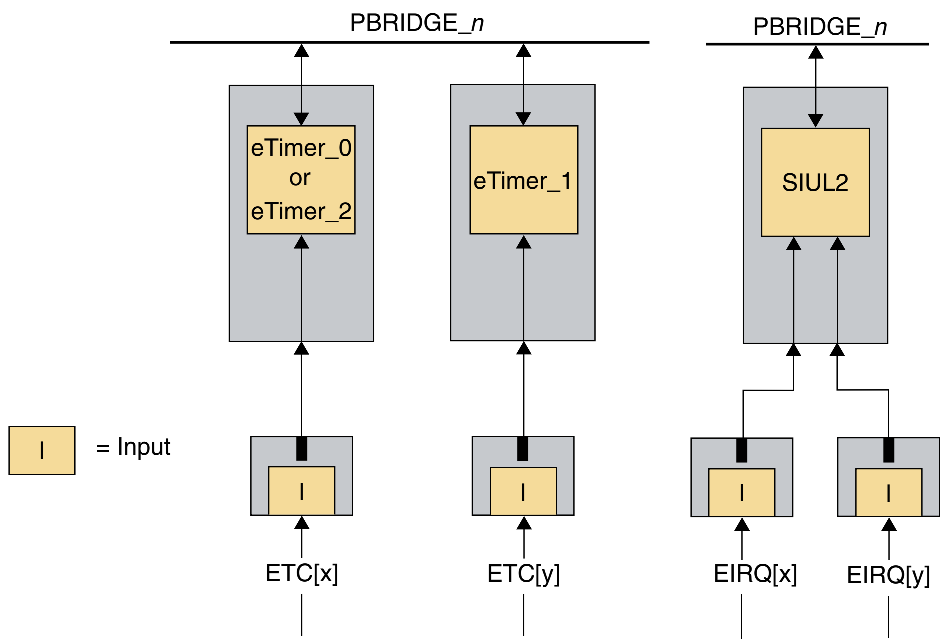


Figure 5-4. Double read encoder input (IRQ triggered)

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

5.7.2.1.2 Software

Digital inputs used for functional safety purposes are assumed to be input redundantly as described in this section. The table below lists two element safety functions for input in the 'Function' column, the corresponding safety integrity functions in the 'Test' column and their execution frequency. Alternative solutions with sufficient diagnostic coverage are possible in the 'Frequency' column.

Table 5-3. Digital inputs software tests

Function	Test	Frequency
Double Read Digital Inputs	SIUL_SWTEST_REGCRC	Once after programming
	GPI_SWTEST_CMP	Once for every acquisition
Double Read PWM Inputs	ETIMER0_SWTEST_REGCRC	Once after programming
	ETIMER1_SWTEST_REGCRC	Once after programming
	SIUL_SWTEST_REGCRC	Once after programming
	ETIMERI_SWTEST_CMP	Once for every acquisition

5.7.2.1.2.1 Double read digital inputs

Rationale: To check that the configuration of the two I/Os used correspond with the expected configuration, to reduce the likelihood of CMF caused by incorrectly configured I/Os, and to check that the two input values read are similar.

Implementation hint: Functional safety integrity is achieved by replicated reading and software comparison by the processing function. The application can implement the tests SIUL_SWTEST_REGCRC and GPI_SWTEST_CMP.

5.7.2.1.2.1.1 Implementation details

The only hardware element that can be used for the safety function is the general purpose input/output (GPIO).

Implementation hint: Every I/O that is not dedicated to a single function can be configured as GPIO. I/Os that are dedicated to ADC are an exception to this rule, as they can only be configured as inputs.

Note

> **Caution:** Redundant GPIO should be selected in a way that their signals are not adjacent, which helps minimize the likelihood of CMFs.

5.7.2.1.2.1.2 SIUL_SWTEST_REGCRC

For implementation details of <*module*>_SWTEST_REGCRC functions, refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.1.2.1.3 GPI_SWTEST_CMP

This software test is used to execute the comparison between the double reads performed by the independent channels. It reads the outputs sequentially. This allows any GPIO to be used, but could result in a wrong result if the state of the input changes between reading the first and second inputs.

An alternative implementation would be to use the parallel data input registers (PGPDI) in the same way that the GPODW_SWAPP_WRITE uses the output equivalent of these registers. This would allow the inputs to be read at the same point in time but would restrict the GPIO that could be used.

5.7.2.1.2.2 Double read PWM inputs

This approach reads two PWM inputs in parallel using two eTimers, then compares the results.

Rationale: To check that the configuration of the modules used by this safety function compare to the expected configuration and to validate that the two sets of read data correlate.

Implementation hint: The software tests that the application may implement are:

- ETIMER0_SWTEST_REGCRC
- ETIMER1_SWTEST_REGCRC
- SIUL_SWTEST_REGCRC

In addition, the double reads shall be compared by the application with the implementation of the following test:

- ETIMERI_SWTEST_CMP.

The SIUL2 module may be configured (via the appropriate SIUL_MSCR*n* and SIUL_IMCR*n*) to provide configuration and input direction of the input GPIO.

5.7.2.1.2.2.1 Implementation details

Rationale: To reduce the risk of cascading faults due to shared resources.

Implementation hint: The following hardware elements shall be used for the safety function:

- eTimer_0 channels
- eTimer_1 channels

The system integrator may select one channel from the eTimer_0 module and another from the eTimer_1.

5.7.2.1.2.2.2 ETIMERx_SWTEST_REGCRC and SIUL_SWTEST_REGCRC

These functions check the correct configuration of all involved modules. See Cyclic Redundancy Checker Unit (CRC) for implementation of the <*module*>_SWTEST_REGCRC functions.

5.7.2.1.2.2.3 ETIMERI_SWTEST_CMP

This test is used to execute the comparison between the double reads of PWM inputs performed by two channels of different eTimer. The comparison may take into account possible approximation because of different capturing of the asynchronous input signals.

5.7.2.1.2.3 Synchronize sequential read input

The synchronize sequential read inputs is implemented by the CTU, which generates the trigger for events according to the triggered mode or the sequential mode. The CTU can be used if the synchronization of the reading of some inputs with some events is required. The following mix of hardware mechanisms and software safety integrity measures implemented at the application level provides respective functional safety integrity . Depending on the assignment of inputs to CTU(s) also the existence of two CTUs can be used to fulfill the requirement:

- CTU_HWSWTEST_TRIGGERNUM
- CTU_SWTEST_TRIGGERTIME
- CTU_HWSWTEST_TRIGGEROVERRUN
- CTU_HWSWTEST_ADCCOMMAND (only if the input is an analog signal)
- CTU_SWTEST_ETIMERCOMMAND
- CTU_HW_CFGINTEGRITY

5.7.2.1.2.3.1 Hardware element

The synchronize sequential read input is implemented by the CTU, which generates the trigger events according to one of the two operation modes shown in Figure 5-5.

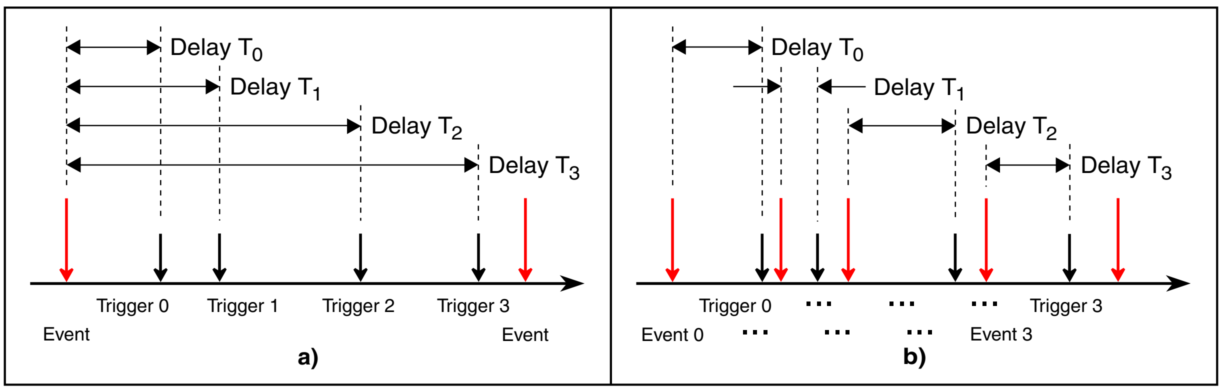


Figure 5-5. CTU operating modes: a) triggered and b) sequential

The CTU receives various incoming signals (Event X in Figure 5-5) from different sources. These signals are then processed to generate trigger events (Trigger X in Figure 5-5). An event can be a rising edge, a falling edge or both edges of each incoming signal. The output trigger can be a pulse, an ADC command (or a stream of consecutive commands) or both to one or more peripherals (for example, ADC, eTimers, and so on).

In triggered mode, the input event, which can be also a combination (logical OR) of several signals, determines the reload/restart of the CTU counter and up to eight comparators are available to generate up to eight output triggers with a given delay with respect to the reload signal. In sequential mode, one comparator can be used to generate a trigger with a given delay with respect to one out of eight input events (Event 0 works as the reload event).

Implementation hint: The CTU is configured so that the output triggers are generated with the desired time schedule with respect to the input event(s).

For each trigger the set of ADC commands and pulses to be generated are defined.

Particularly, each ADC command specifies which channel is acquired by which ADC, if two ADCs perform a concurrent conversion or just one of them is operational, and in which CTU internal FIFO the result(s) will be stored. Four FIFOs are available ($2 \times 16 + 2 \times 4$). In case of a concurrent acquisition the same FIFO is used for both results. ADCs are configured to accept commands from the CTU (instead of commands provided via software). Multiple single or concurrent acquisitions can be scheduled for each trigger events (overall 24 commands per control period, for example, between two successive reload signals). The next command is sent when the ADC signals the completion of previous acquisition.

Recommendation: The CTU can be configured to generate interrupt requests when a trigger occurs (for example, to trigger READ DIGITAL INPUTS).

5.7.2.1.2.3.2 Implementation details

The following hardware elements may be used for the safety function:

- CTU
- One eTimer channel
- Another eTimer channel

Table 5-4. CTU software tests

Function	Test	Frequency
Synchronize sequential read input	CTU_HWSWTEST_TRIGGERNUM	Once for every control period (< FTTI)

Table continues on the next page...

Table 5-4. CTU software tests (continued)

Function	Test	Frequency
	CTU_SWTEST_TRIGGERTIME	Once for every CTU control period (triggered mode) or every trigger (sequential mode)
	CTU_HWSWTEST_TRIGGEROVERRUN	Once for every trigger
	CTU_HWSWTEST_ADCCOMMAND	Once for every ADC command
	CTU_SWTEST_ETIMERCOMMAND	Once for every control period (< FTTI)
	CTU_HW_CFGINTEGRITY	Once for every control period (< FTTI)

5.7.2.2 Digital outputs

Functional safety digital outputs are always assumed to be written either redundantly or with read back. In case of single output with read back, the feedback loop should be as large as possible to cover faults on system level also. The figure below depicts the connection of two (functional safety critical) actuators connected to the MPC5744P. Actuator 1 is connected to an output peripheral, for example, a motor is connected to a PWM output (output peripheral 3). The signal generated by the output peripheral 3 can be input to an input peripheral, for example, an eTimer. This measure is to confirm, that the generated output signal is correct. This read back may be internally of the MPC5744P (internal read back) or externally (external read back). The external read back covers more types of failures (for example, corrupt wire bonds or solder joints) than the internal read back, but still does not guarantee, that the actuator really behaves as desired. This is achieved by including the actuator and sensor into the read back loop. An alternative solution is to redundantly output a signal. For example, actuator 2 consists of two relays in series to switch off a functional safety-relevant supply voltage. The selection of the suited output connection is part of the I/O functional safety concept on system level.

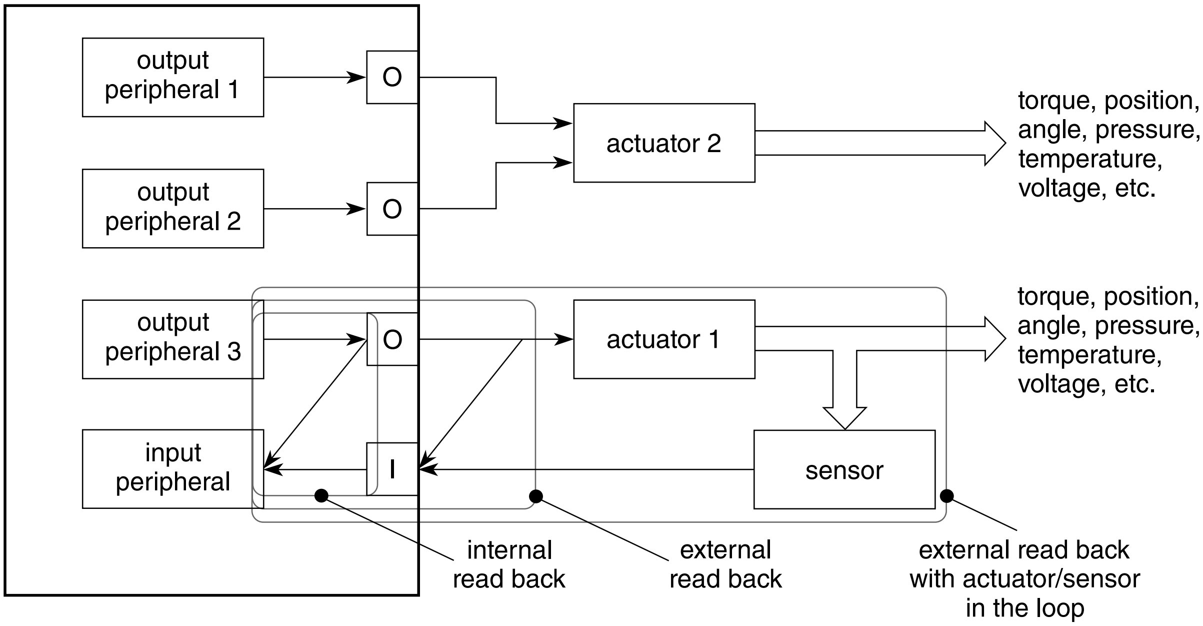


Figure 5-6. Digital Outputs with redundancy and read back

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

5.7.2.2.1 Hardware

5.7.2.2.1.1 Single write digital output

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

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

- Single Write Digital Output with internal read-back [4] (Figure 5-7, right):

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

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

4. Internal read back does not cover package faults (for example, wire bond, etc.).

This procedure output compares the PWM read-back provided by a single channel of the eTimer_0 (eTimer_1, eTimer_2) with the expected values that have been written to the external pad of the FlexPWM1 (FlexPWM_0) output channel.

- Single Write PWM Output with internal read-back [4] (Figure 5-8, right):

This procedure output compares the PWM read-back by a single channel of the eTimer_0 (eTimer_1, eTimer_2) with the expected values that have been written to the FlexPWM1 (FlexPWM_0) output channel.

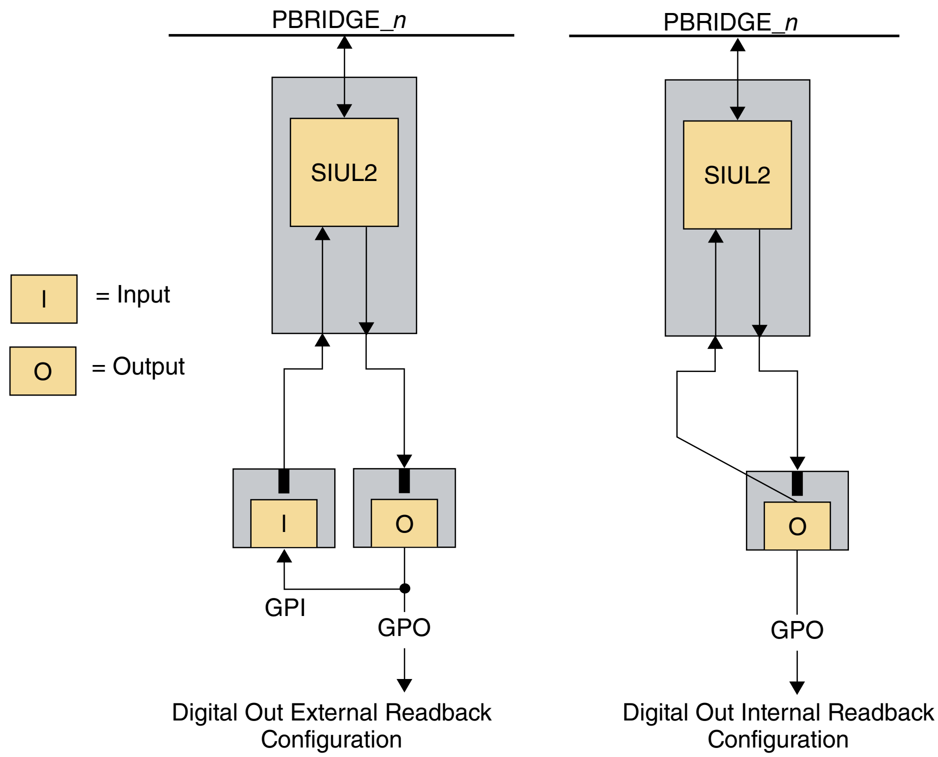


Figure 5-7. Single write digital output with read-back

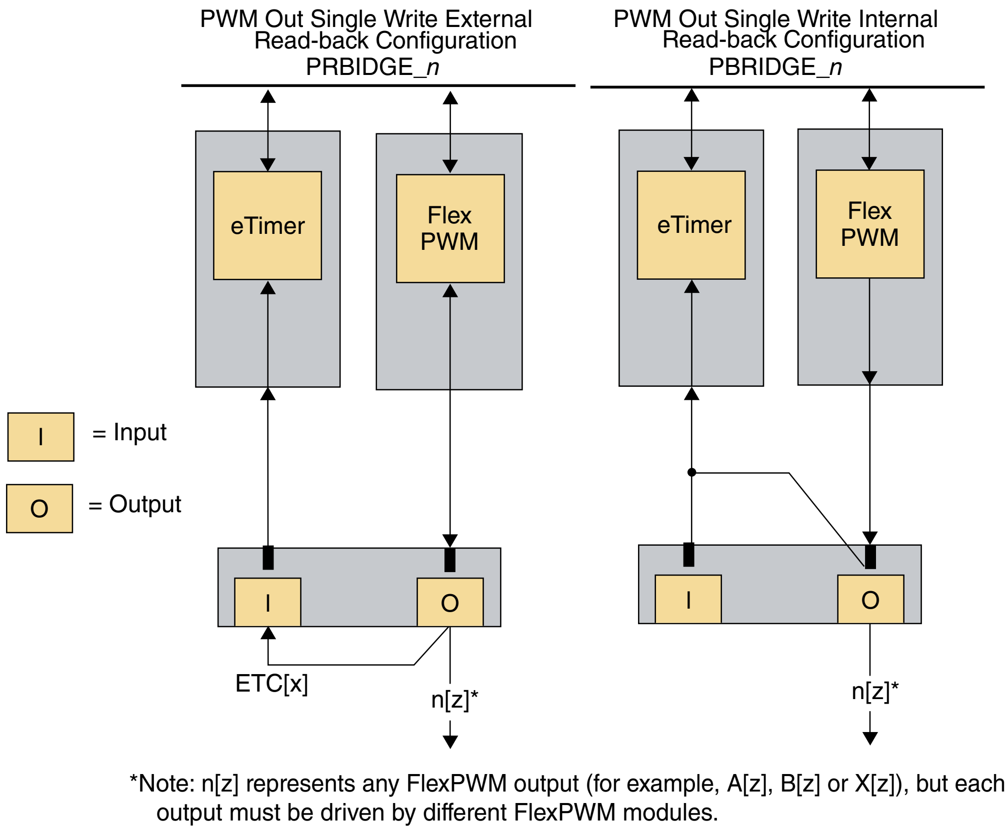


Figure 5-8. Single write PWM output with read-back

5.7.2.2.1.2 Double write digital output

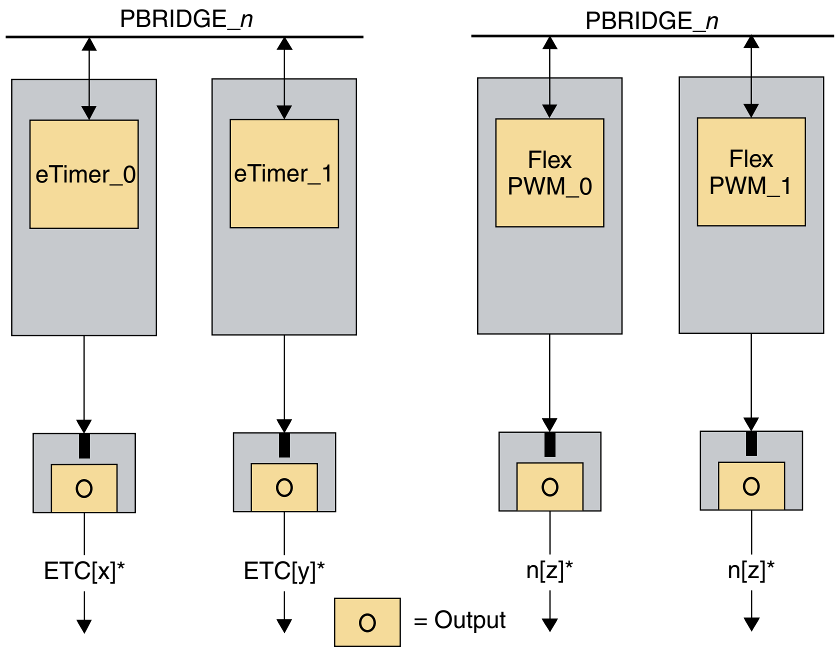
- Double write digital output

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

- Double write PWM output

  - The hardware elements are used to perform a double write PWM output:

    - eTimer_0 and eTimer_1
    - eTimer_1 and eTimer_2
    - FlexPWM_0 and FlexPWM_1



Note: n[z] represents any FlexPWM output (for example, A[z], B[z] or X[z]), but each output must be driven by different FlexPWM modules. The same consideration is valid for the eTimer; any eTimer output may be used, but each output must be driven by different eTimer module.

Figure 5-9. Double write PWM output

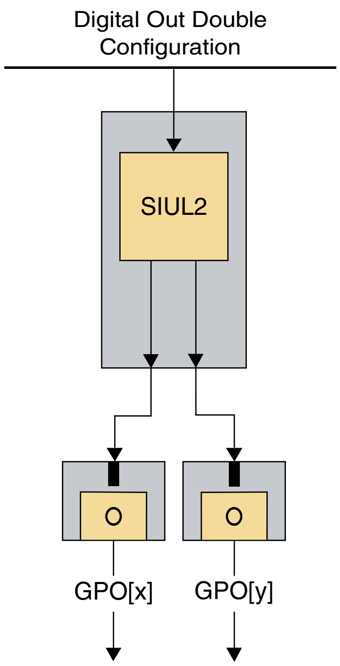


Figure 5-10. Double write digital output

5.7.2.2.2 Software

Digital outputs used for functional safety purposes are assumed to be written either redundantly or with read back as described in this section. Table 5-5 lists four element safety functions for output, the corresponding safety integrity functions and their execution frequency. Alternative solutions with sufficient diagnostic coverage are possible.

Table 5-5. Digital outputs software tests

Function	Test	Frequency
Single Write Digital Outputs With Read Back	SIUL_SWTEST_REGCRC	Once after programming
	GPOERB_SWTEST_CMP	Once per FTTI
	GPOIRB_SWTEST_CMP	Once per FTTI
Double Write Digital Outputs	SIUL_SWTEST_REGCRC	Once after programming
	GPODW_SWAPP_WRITE	Once per FTTI
Single Write PWM Outputs With Read Back	SIUL_SWTEST_REGCRC	Once after programming
	ETIMER0_SWTEST_REGCRC[1]	Once after programming
	ETIMER1_SWTEST_REGCRC1	Once after programming
	ETIMER2_SWTEST_REGCRC1	Once after programming
	FLEXPWM0_SWTEST_REGCRC[2]	Once after programming
	FLEXPWM1_SWTEST_REGCRC2	Once after programming
	PWMRB_SWTEST_CMP	Once per FTTI
Double Write PWM Outputs	SIUL_SWTEST_REGCRC	Once after programming[3]
	ETIMER0_SWTEST_REGCRC1	Once after programming
	ETIMER1_SWTEST_REGCRC1	Once after programming
	ETIMER2_SWTEST_REGCRC1	Once after programming
	FLEXPWM0_SWTEST_REGCRC2	Once after programming
	FLEXPWM1_SWTEST_REGCRC2	Once after programming
	PWMDW_SWAPP_WRITE	Once per FTTI

1. This test is required only if the eTimer channels are used for the safety function.
2. This test is required only if the FlexPWM channels are used for the safety function.
3. If a change in a single SIUL2 configuration register is capable of affecting both the output and the read-back paths, then SIUL_SWTEST_REGCRC may be executed every FTTI. In all other cases configuration errors are covered by the software comparison.

5.7.2.2.2.1 Single write digital outputs with read-back

The SIUL2 hardware element is used to perform a Single Write Digital Output With Read-Back (see Figure 5-7).

Rationale: To check whether written data is coherent to the expected data

Implementation hint: The read back may be implemented either with external or with internal readback.

The SIUL2 element is correctly configured to provide the output write and the pad directions as follows:

- External read back – SIUL2 is configured to read back the signal via an additional pad, and the loopback is performed outside the device. In this configuration, only half of the available digital outputs are available as functional safety outputs.
- Internal read back – SIUL2 is configured to read back the pad value via an internal read path. All pads dedicated to digital input/output are capable of reading the pad digital status using the input logic.

Rationale: To reduce the likelihood of a CMF caused by incorrect configuration of pads

Implementation hint: The application software integrates software test SIUL_SWTEST_REGCRC in the application to check the correct configuration of the pads, and to compare a read back with the digital output write. GPOERB_SWTEST_CMP may be used for the external read back or GPOIRB_SWTEST_CMP for internal read back.

5.7.2.2.2.1.1 Implementation details

The SIUL2 hardware element may be used for the safety function.

Note

> Pads that are not dedicated to a single function can be configured as GPIO. Pads dedicated to ADC are an exception to this rule, as they can only be configured as inputs.

5.7.2.2.2.1.2 SIUL_SWTEST_REGCRC

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.2.2.1.3 GPOERB_SWTEST_CMP

This software test is used to execute the comparison between the desired output values and the value read back via external read back configuration. After writing the output value, the test reads the status of the digital input.

Rationale: To check if the read data equals the written data

Implementation hint: The output is externally (on system level) connected to an input I/O. After writing the value to the output signal, the input is read to check that the correct output is present.

5.7.2.2.2.1.4 GPOIRB_SWTEST_CMP

Rationale: To check if the read data equals the written data.

This software test is used to execute the comparison between the desired output values and the value read back via internal read back configuration. After writing the output value, the test reads the status.

5.7.2.2.2.2 Double write digital outputs

The SIUL2 hardware element is used to perform a Double Write Digital Output.

Rationale: To configure pads used by this safety function and reduce the likelihood of a CMF caused by incorrect configuration of pads

Implementation hint: The SIUL2 is configured by application software to correctly define the configuration of the outputs used. The software performs a double write.

Rationale: To reduce the likelihood of a CMF caused by incorrect configuration of the pads

Implementation hint: To achieve the integrity of the two output channels, the application validates the SIUL2 configuration implementing the SIUL_SWTEST_REGCRC.

Rationale: To write a digital output by exploiting redundancy

Implementation hint: The application software implements the double output write as defined by the GPODW_SWAPP_WRITE.

5.7.2.2.2.2.1 Implementation details

The only hardware element that can be used for the safety function is the GPIO.

Every pad not dedicated to a single function may be configured as GPIO. Pads dedicated to ADC are an exception to this rule, as they can be configured as inputs only.

5.7.2.2.2.2.2 GPODW_SWAPP_WRITE

Rationale: To prevent SPFs in the SIUL2 to influence the actuator control in a dangerous way

Implementation hint: The output write of a redundant channel may be implemented by writing the two outputs with a single instruction to the appropriate register and this register may be checked by read back.

To write two or more GPIOs with a single instruction, the Masked Parallel GPIO Pad Data Out register (SIUL_MPGPDO*n*) register can be used. The two GPIOs used must be in the same SIUL_MPGPDO*n* register.

To protect the value of the other GPIOs that belong to the same SIUL_MPGPDO*n*, the MASK field of the SIUL_MPGPDO*n* register needs to be properly configured.

When using a single write (atomic) instruction to SIUL_MPGPDO*n* register, it is good practice to read back (read after write) the register content due to the fact that a transient fault in the SIUL2 IPS interface can affect in principle both output channels. The readback is needed to cover this common mode of failure. An alternative implementation would be to write the two outputs separately not using the parallel register, resulting in a small delay in output change between the channels.

5.7.2.2.2.3 Single write PWM outputs with read-back

The following combination of elements may be used to perform a Write PWM Output With Read-Back:

- eTimer_0 – FlexPWM_0
- eTimer_1 – FlexPWM_1
- eTimer_2 – FlexPWM_0

These units shall be configured to implement one PWM output channel and (via internal read-back) the eTimer_0 input PWM channel. The SIUL2 shall be configured to define the configuration of the output pads used. The software performs a write operation followed by a read operation. To achieve the integrity of the two output channels, the application shall tests the SIUL2 configuration implementing the SIUL_SWTEST_REGCRC (to reduce the likelihood of a CMF caused by incorrect configuration of the pads).

Rationale: To check that the configuration of the modules used by this safety function adheres to the expected configuration.

Implementation hint: A single channel of the eTimer is used with a multiplexing of the internal read-back of the different output of the FlexPWM. The read-back paths are limited to six signals, two for each sub-module of the FlexPWM.

The following tests validate the correct configuration for the eTimer(s):

- FLEXPWM0_SWTEST_REGCRC
- FLEXPWM1_SWTEST_REGCRC

- ETIMER0_SWTEST_REGCRC
- ETIMER1_SWTEST_REGCRC
- ETIMER2_SWTEST_REGCRC

Rationale: To check that the written data is what is expected.

Implementation hint: The application software writes to the output port and then compare the written value via the read-back (PWMRB_SWTEST_CMP).

5.7.2.2.2.3.1 Implementation details

The following hardware elements may be used for the safety function:

- eTimer_0 channels
- eTimer_1 channels
- eTimer_2 channels
- FlexPWM_0 channels
- FlexPWM_1 channels

5.7.2.2.2.3.2 FLEXPWMx_SWTEST_REGCRC and ETIMERx_SWTEST_REGCRC

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.2.2.3.3 PWMRB_SWTEST_CMP

This test compares the PWM read back provided by a single channel of the eTimer_1 (eTimer_0) with the expected values that have been written to the FlexPWM_0 (FlexPWM_1) output channel.

For this test, FlexPWM_0 is used to generate a PWM output and eTimer_1 is used to read back and verify the output. Another combination could be used if required in an application.

5.7.2.2.2.4 Double write PWM outputs

Rationale: The hardware elements eTimer_0, eTimer_1, eTimer_2, FlexPWM_0 and FlexPWM_1 are used to perform a double Write PWM Output.

Implementation hint: These units are configured to implement two independent PWM channels. The SIUL2 is configured to define the configuration of the output pads used. The software performs a double write (see PWMDW_SWAPP_WRITE).

Rationale: To reduce the risk of CCF due to spatial proximity

Implementation hint: Using adjacent pads as redundant I/O increases the likelihood of CMFs. Therefore, it is preferable to use I/O that do not share the same configuration and data registers in the SIUL2.

Rationale: To reduce the likelihood of a CMF caused by incorrect configuration of the pads

Implementation hint: To improve the integrity of the two output channels, the application should test the SIUL2 configuration implementing the SIUL_SWTEST_REGCRC.

Rationale: To check that the configuration of the modules used by this safety function adhere to the expected configuration

Implementation hint: The application software shall implement a test for the register configuration:

- ETIMER0_SWTEST_REGCRC (for eTimer)
- ETIMER1_SWTEST_REGCRC (for eTimer)
- ETIMER2_SWTEST_REGCRC (for eTimer)
- FLEXPWM0_SWTEST_REGCRC (for FlexPWM)
- FLEXPWM1_SWTEST_REGCRC (for FlexPWM)

Rationale: To reduce the possibility of cascading a failure to shared circuitries, different modules should be used.

Implementation hint: The output write of a redundant PWM channel is implemented by writing the new output values to both PWM channels. The system integrator can decide whether to use two of the three eTimers (eTimer_0, eTimer_1, eTimer_2) or one of the two FlexPWMs (FlexPWM_0, FlexPWM_1).

5.7.2.2.2.4.1 Implementation details

The following hardware elements are used for the safety function:

- eTimer_0 channels
- eTimer_1 channels

- eTimer_2 channels
- FlexPWM_0 channels
- FlexPWM_1 channels

5.7.2.2.2.4.2 SIUL_SWTEST_REGCRC

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.2.2.4.3 PWMDW_SWAPP_WRITE

If the content of the PWM outputs are changed, care must be taken since the outputs can not be updated synchronously. Therefore for a short period of time both outputs could be different.

5.7.2.3 Analog inputs

5.7.2.3.1 Hardware

Two options for reading analog inputs exist:

- Single Read Analog Inputs
- Double Read Analog Inputs

Apart from BISTs described in Analog to Digital Converter (ADC), additional tests may be implemented in software as described in section Single read analog inputs and Double Read Analog Inputs.

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

5.7.2.3.1.1 Single read analog inputs

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

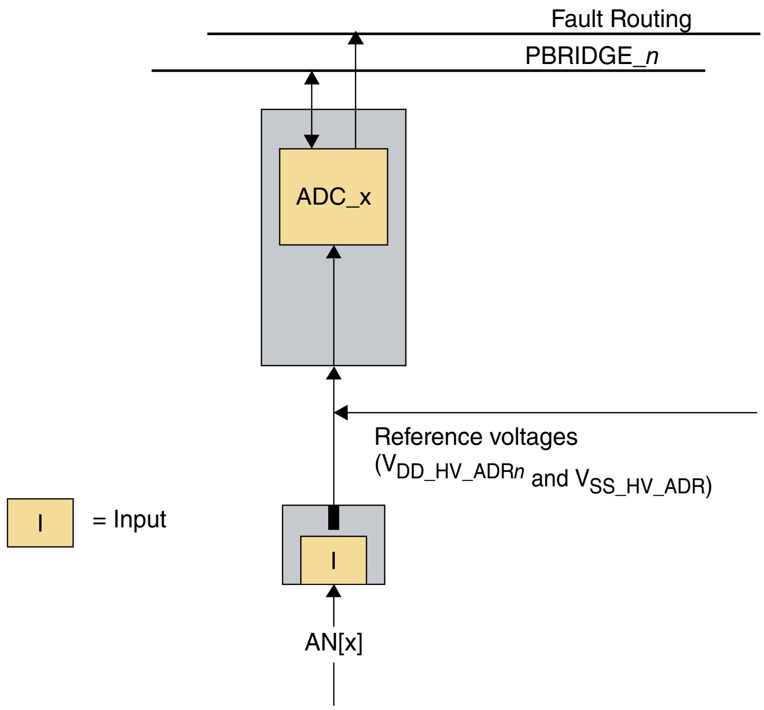


Figure 5-11. Single read analog input configuration

5.7.2.3.1.2 Double read analog inputs

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

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

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

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

Implementation hint: One shared ADC-channel may not be used for both inputs of the Double Read Analog Input function.

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

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

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

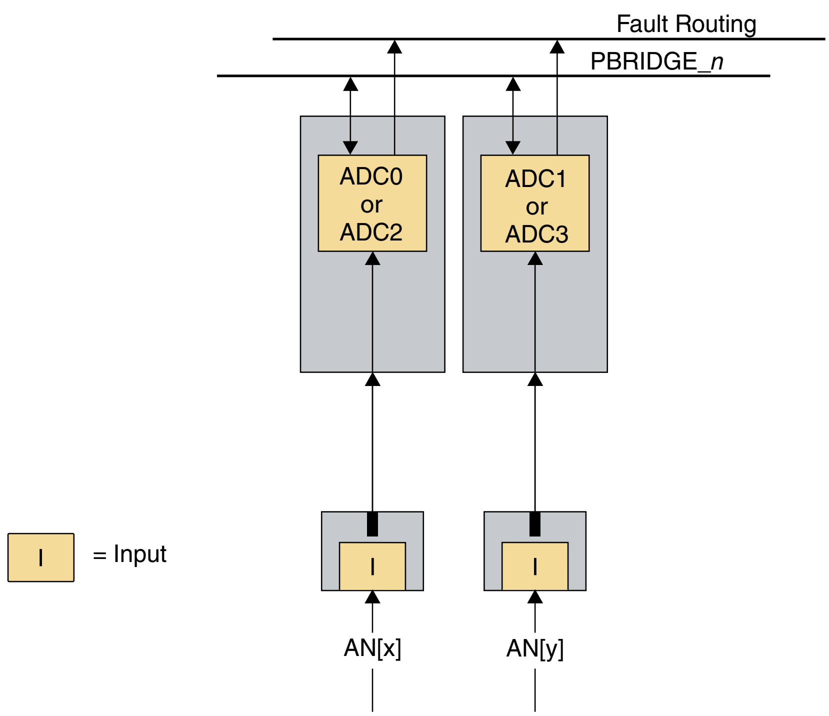


Figure 5-12. Double read analog inputs configuration

5.7.2.3.2 Software

Analog inputs used for functional safety purposes are assumed to be input redundantly as described in this section. Table 5-6 list two element safety functions for analog input, the corresponding safety integrity functions and their execution frequency. Alternative solutions with sufficient diagnostic coverage are possible.

Table 5-6. Analog inputs software tests

Function	Test	Frequency
Single Read Analog Inputs	SUPPLY SELF-TEST	Once in the FTTI
	CAPACITIVE SELF-TEST	Once in the FTTI
	ADC_SWTEST_TEST1	Once in the FTTI
	ADC_SWTEST_TEST2	Once in the FTTI
	ADC_SWTEST_VALCHK	Once for every acquisition
	ADC_SWTEST_OVERSAMPLING	Once for every acquisition
	ADC0_SWTEST_REGCRC	Once in the FTTI
	ADC1_SWTEST_REGCRC	Once in the FTTI
	ADC2_SWTEST_REGCRC	Once in the FTTI
	ADC3_SWTEST_REGCRC	Once in the FTTI
	SIUL_SWTEST_REGCRC	Once in the FTTI
Double Read Analog Inputs	SUPPLY SELF-TEST	Once after boot
	CAPACITIVE SELF-TEST	Once after boot
	ADC0_SWTEST_REGCRC	Once after programming
	ADC1_SWTEST_REGCRC	Once after programming
	ADC2_SWTEST_REGCRC	Once after programming
	ADC3_SWTEST_REGCRC	Once after programming
	SIUL_SWTEST_REGCRC	Once after programming
	ADC_SWTEST_CMP	Once for every acquisition

5.7.2.3.2.1 Single read analog inputs

To support a high diagnostic coverage two known reference supply voltages are utilized by two software tests which are described in the following sections (ADC_SWTEST_TEST1 and ADC_SWTEST_TEST2).

The reference supply voltages are the following:

- $V_{DD\_HV\_ADRE0}$ (ADC0/ADC2 high voltage reference)
- $V_{DD\_HV\_ADRE1}$ (ADC1/ADC3 high voltage reference)

- $V_{SS\_HV\_ADRE0}$ (ADC0/ADC2 reference ground)
- $V_{SS\_HV\_ADRE1}$ (ADC1/ADC3 reference ground)

The SIUL2 unit is configured to correctly enable the ADC inputs. The pads used for analog inputs can only be configured as inputs.

Single Read Analog Inputs may be implemented using the following safety integrity functions at the application level:

- ADC_SWTEST_TEST1
- ADC_SWTEST_TEST2
- ADC_SWTEST_VALCHK
- ADC0_SWTEST_REGCRC, ADC1_SWTEST_REGCRC, ADC2_SWTEST_REGCRC, or ADC3_SWTEST_REGCRC
- SIUL_SWTEST_REGCRC
- ADC_SWTEST_OVERSAMPLING

5.7.2.3.2.1.1 Implementation details

The following hardware elements can be used for the safety integrity functions:

- Analog input channels AN[0:8] of ADC0
- Analog input channels AN[11:14] of ADC0 and ADC1 (shared channels)
- Analog input channels AN[0:8] of ADC1
- Analog input channels AN[0:2] of ADC2 and ADC3 (shared channels)
- Analog input channels AN[3:7] of ADC3

5.7.2.3.2.1.2 SIUL_SWTEST_REGCRC

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

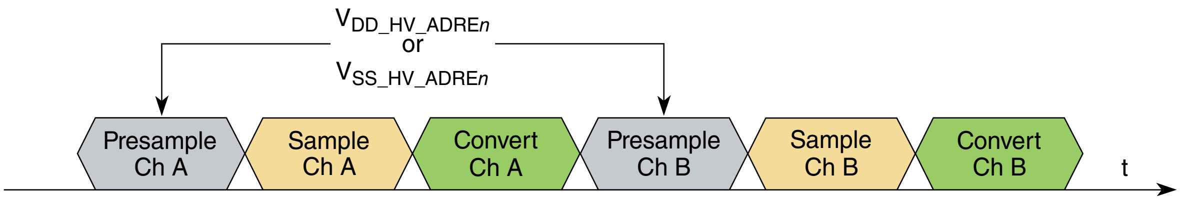
5.7.2.3.2.1.3 ADCn_SWTEST_REGCRC

If ADC_0 is used the ADC0_SWTEST_REGCRC may be used. If ADC_1 is used the ADC1_SWTEST_REGCRC may be used. If ADC_2 is used the ADC2_SWTEST_REGCRC may be used.If ADC_3 is used the ADC3_SWTEST_REGCRC may be used.

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.3.2.1.4 ADC_SWTEST_TEST1 (open detection)

This test exploits the presampling feature of the ADC. Presampling allows to precharge or discharge of the ADC internal capacitor before it starts the sampling and conversion phases of the analog input received from the pads. During the presampling phase, the ADC samples the internally generated voltage. While in the sampling phase, the ADC samples analog input coming from the pads. In the conversion phase, the last sampled value is converted to a digital value. Figure 5-13 shows the normal sequence of operation for two channels (Presampling – Sampling – Conversion).



Note: Either $V_{DD\_HV\_ADR0/1}$ or $V_{SS\_HV\_ADR0/1}$ can be used as presampling voltage.

Figure 5-13. Implementation of ADC_SWTEST_TEST1

Reference voltages, which can be used during presampling phase, is either $V_{DD\_HV\_ADRE}$ or $V_{SS\_HV\_ADRE}$.

If there is an open failure in the analog multiplexing circuitry, the signal converted by the ADC is not the analog input coming from the pad, but the presampling reference voltage ($V_{DD\_HV\_ADRn}$ or $V_{SS\_HV\_ADRn}$). Figure 5-14 depicts the signal path in the analog multiplexing circuitry for presampling phase and conversion phase.

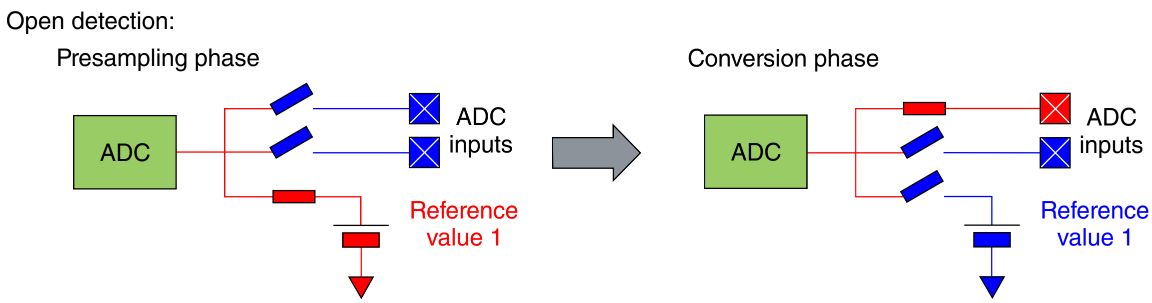


Figure 5-14. ADC_SWTEST_TEST1 (open detection)

Each analog input channel used by the safety function may be tested by system level measures (software).

Since the pads dedicated to analog inputs are of type INPUT, a missing enable from the SIUL2 results in an open failure.

Rationale: To detect open failures of the channel multiplexing circuitry (see Figure 5-14).

Implementation hint: Presampling can be enabled on a per channel basis through the ADC_*n*_PSR0 register. ADC_*n*_PCSR[PREVAL0] selects which reference voltage is used to precharge/discharge the ADC internal capacitor, (ADC_*n*_PSCR[PRECONV] = 0). (See "Analog-to-Digital Converter (ADC)" chapter in the *MPC5744P Reference Manual* for details on the presampling feature).

Note

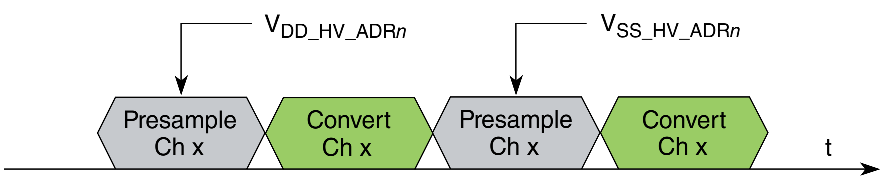
Caution! To reduce the likelihood of a false indication of an open fault in the analog multiplexor, signals connected to the ADC inputs should not be outside of the limits of the reference voltages ($V_{DD\_HV\_ADR}$, $V_{SS\_HV\_ADR}$). In case this limitation cannot be fulfilled by the application, a more complex algorithm may be necessary (for example, run the test three times with $V_{DD\_HV\_ADR}$, $V_{SS\_HV\_ADR}$, $V_{DD\_HV\_ADR}$).

5.7.2.3.2.1.5 ADC_SWTEST_TEST2 (short detection)

To detect short failures two different voltages are acquired by the ADC. If these values are different from the expected ones, a short failure on the multiplexed circuitry has been detected.

To implement this test a presampling feature of the ADC can be exploited. The presampling may be configured in such a way that the sampling of the channel is bypassed and the presampling reference supply voltages are converted.

During the first step the $V_{DD\_HV\_ADRn}$ is converted and compared with the expected value; then the $V_{SS\_HV\_ADRn}$ is converted and compared with the expected one (see Figure 5-15).



Note: Either $V_{DD\_HV\_ADR0/1}$ or $V_{SS\_HV\_ADR0/1}$ can be used as presampling voltage.

Figure 5-15. Implementation of ADC_SWTEST_TEST2

Rationale: To detect short failures of the channel multiplexing circuitry (see Figure 5-16).

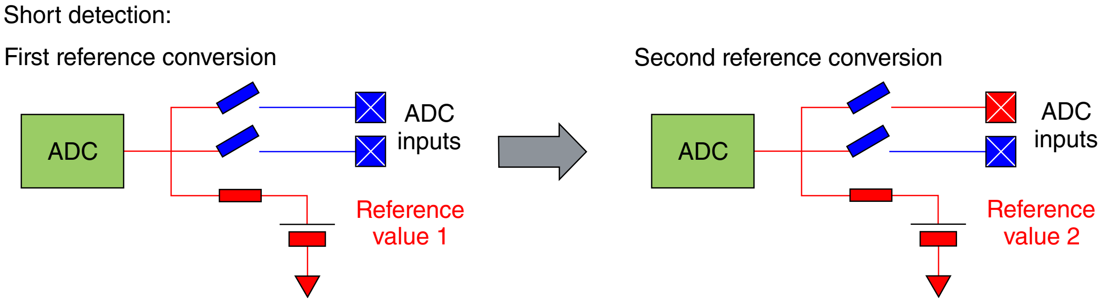


Figure 5-16. ADC_SWTEST_TEST2 (short detection)

Implementation hint: Presampling can be enabled on a per channel basis through the ADC*n*_PSR0 register. ADC*n*_PCSR[PREVAL0] selects which reference voltage is used to precharge/discharge the ADC internal capacitor. To bypass the conversion of the input channel and convert the presampled values, ADC*n*_PCSR[PRECONV] = 1. (See "Analog-to-Digital Converter (ADC)" chapter in the *MPC5744P Reference Manual* for details on the presampling feature).

5.7.2.3.2.1.6 ADC_SWTEST_VALCHK

When ADC conversion is triggered by the CTU, the acquired digital sample data are stored into a dual queue along with information about the channel that performed the acquisition. The checking of the expected channel provides coverage of the control logic and part of the queue logic. Checking of the expected sequence of acquired channels provides the coverage of the control logic and part of the queue logic.

The goal of this software test is to verify correct operation of the control and queue logic of the ADC, and also the CTU, if used. The way this software measure is implemented depends on how the ADC is configured (for example, CTU or CPU mode):

- When the ADC is used in CPU mode, the acquired value is read by the ADC_CDR*n*. This register includes ADC_CDR*n*[VALID] and ADC_CDR*n*[RESULT] fields as well as channel n converted data (ADC_CDR*n*[CDATA]). These fields provide status information about the data acquisition. Application software should read and verify these fields after every acquisition.
- When ADC conversion is triggered by the CTU, the acquired digital sample data is stored into a dual queue along with information about the channel that performed the acquisition. The checking of the expected channel provides coverage of the control logic and part of the queue logic.

Implementation hint: If ADC is configured to work in CTU mode, the conversion results are stored in CTU FIFOs (see the 'Cross-Triggering Unit (CTU)' chapter in the *MPC5744P Reference Manual* for details). Along with the converted data, the converted channel number and ADC module are stored. CTU includes two sets of registers to read this information (FIFO Right aligned data, FR*x*, and FIFO Right aligned data, FL*x*). These registers may be read to check that the sequence of the acquired channel is what is expected.

5.7.2.3.2.1.7 ADC_SWTEST_OVERSAMPLING

During Single Read Analog Inputs, the ADC_SWTEST_OVERSAMPLING may be implemented as counter measure against random fault.

ADC_SWTEST_OVERSAMPLING is an acquisition redundant in time.

It refers to sampling the signal at a rate significantly higher than the Nyquist frequency related to the input signal. If there is a fault, the acquired values will not be correlated. This safety integrity measure compares the acquired value to check the correlation.

Against a random fault, three consecutive analog values are converted for each acquisition to implement the ADC_SWTEST_OVERSAMPLING. The figure below shows the sampling of an analog signal at different points in time (A_1, A_2 and A_3). Every conversion is indicated by an arrow, which indicates the converted digital value by its length. The second acquisition (A_2) is faulty because the first converted value is quite different respect the other two.

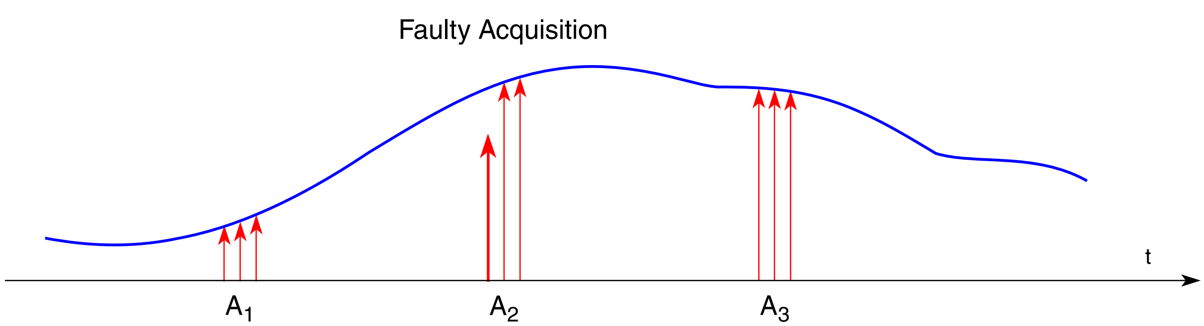


Figure 5-17. Series of acquired analog values

5.7.2.3.2.2 Double Read Analog Inputs

Rationale: To validate that the configuration of the modules used by this safety function corresponds with what is expected. To reduce the likelihood of CMFs caused by improper configuration of the pads.

Implementation hint: Double Read Analog Inputs may be implemented using the following safety integrity functions at the application level:

- ADC0_SWTEST_REGCRC
- ADC1_SWTEST_REGCRC
- ADC2_SWTEST_REGCRC
- ADC3_SWTEST_REGCRC
- SIUL_SWTEST_REGCRC

Rationale: To validate that the two sets of read data correlate.

Implementation hint: Double Read Analog Inputs may be implemented using the software test ADC_SWTEST_CMP to compare the channel reads.

5.7.2.3.2.2.1 Implementation details

The following hardware elements may be used for the safety function:

- Analog input channels AN[0:8] of ADC_0
- Analog input channels AN[0:8] of ADC_1
- Analog input channels AN[0:2] of ADC_2
- Analog input channels AN[0:7] of ADC_3

One channel from different ADC modules may be used, for example, one from ADC0 and one from the ADC1.

5.7.2.3.2.2.2 SIUL_SWTEST_REGCRC

For implementation of a <*module*>_SWTEST_REGCRC function please refer to Cyclic Redundancy Checker Unit (CRC).

5.7.2.3.2.2.3 ADC_SWTEST_CMP

This software test is used to execute the comparison between the double reads performed by any combination of two ADC*n* module channels. The comparison may take possible conversion tolerances into account.

5.7.2.4 Other requirements

Rationale: To detect missing eTimer acquisition.

Implementation hint: In the eTimer module, the capture flag (eTimer_n_STS[ICFn]) may be used.

Rationale: To detect stalled quadrature counting.

Implementation hint: When using the eTimer counter to decode a primary and secondary external input as quadrature encoded signals, the eTimer watchdog may be used (see the "Counting Modes" section of the MPC5744P Reference Manual). eTimer watchdog is only available for channel 0.

Implementation hint:

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

5.7.3 PBRIDGE protection

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

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

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

5.7.3.1 Initial checks and configurations

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

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

5.7.4 Temperature Sensors (TSENS)

The MPC5744P has two temperature sensors that are read from the ADC modules (TSENS0 from ADC0 channel 15, TSENS1 from ADC1 channel 15), and the redundant temperature sensors are in separate safety lakes. Each temperature sensor generates one analog voltage which is proportional to the absolute current junction temperature of the device and three digital outputs that signal whether the junction temperature has reached either a preset low temperature threshold or one of two preset high temperature thresholds.

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

- FCCU failure generation according to the defined low and high temperature points

Recommendation: To reduce the likelihood of CMFs related to the effects of temperature threshold violations (for example, due to random hardware faults), the faults may be controlled at the system level.

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.

Implementation hint: When the temperature sensor is enabled, the temperature sensor monitors the internal junction temperature of the chip and asserts a signal if any of the specified temperature thresholds are crossed (see the Temperature threshold detection (digital output generation) section in the Temperature Sensor chapter of the MPC5744P Reference Manual).

Note

> The over and under temperature thresholds can be adjusted by configuring the 'Temperature detector configuration register (PMC_CTL_TD)' located in the PMC module.

Two temperature sensors monitor the substrate temperature to detect over-temperature conditions and under-temperature conditions before they cause any CMFs (for example, faults due to over-temperature which causes identical erroneous results from both cores). The maximum operating junction temperature is specified in the *MPC5744P Data Sheet*. The sensor analog output is forwarded to the appropriate ADC channels for measurement conversion.

5.7.4.1 Initial checks and configurations

Recommendation: If using the temperature sensors as a common mode fault measure during or after initialization, but before executing any safety function, the temperature sensors should be read by software to determine if temperatures are reasonable and within correct operating temperature range.

However, nothing prohibits reading the temperature sensor during execution of the safety function (application run time).

Rationale: A means of assessing functionality of the temperature sensor

Assumption: [SM_066] During power up, the two temperature sensors are to be read by software (TSENS0 from ADC0 channel 15, TSENS1 from ADC1 channel 15). The software must verify that the conversion values are similar, as this is a means of assessing that the sensors are working properly. [end]

Assumption: [SM_064] Application software shall configure the FCCU and the PMC registers related to temperature sensor configuration to react to over-temperature faults of the temperature sensors. [end]

Recommendation: If using the internal temperature sensors and an external temperature sensor as common mode fault measure, improving CMF robustness, the temperature reading from the external sensor should not use the same analog to digital converter (ADC) as TSENS*n*. [end]

5.7.5 Analog to Digital Converter (ADC)

Parts of the Successive Approximation Register (SAR) Analog-to-Digital Converter (ADC) of the MPC5744P do not provide the functional safety integrity to achieve high functional safety integrity targets. Therefore, system level measures are required.

5.7.5.1 Initial checks and configurations

Assumption under certain conditions: [SM_130] When Analog-to-Digital Converter (ADC) of the MPC5744P 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: Once per L-FTTI, the following hardware BISTs of one or both ADC modules 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

These tests can be executed in either of the following modes:

- CPU mode
- CTU mode

Calibration needs to be completed after destructive reset.

In CPU mode, the application software takes care of the hardware self-test activation and checks the test flow and the timing.

In CTU mode, the CTU module takes care of the hardware self-test activation, flow monitoring, and timing. It is important to note that in this operating mode, the CPU does not take part in running the hardware self-test.

Hardware self-tests use analog watchdogs to check the outcome of self-test conversions. The reference thresholds of these watchdogs are saved in the flash memory test sector.

Assumption under certain conditions: [SM_131] Before running the ADC hardware self-test, the system software must copy the reference thresholds from UTest flash memory to the watchdog registers (STAW*n*R). [end]

Rationale: To set the correct threshold for the self-tests.

Implementation hint: To pass the self-test in a noisy environment where the application can live with the noise level, the thresholds that you specify in the STAW4R and STAW5R registers can be relaxed to ±32. However, it is preferable that the environment is cleaned up to match the expectation for better operation instead of threshold relaxation. Relaxing the thresholds more than ±32 can lead to a false pass and is not recommended for safety applications.

NOTE

See the "Self-test analog watchdog" section of the "Analog-to-Digital Converter (ADC)" chapter and the "UTEST flash memory map" table in the "Memory map" chapter of the *MPC5744P Reference Manual* and the application note titled *MPC574xP ADC Self Test* (document AN5015) for more information.

Assumption under certain conditions:[SM_132] When using integrated self-test as the functional safety integrity measure, the analog watchdog for CPU and CTU modes must be enabled for the self-test. The programmable watchdog timeout is smaller than the FTTI. [end]

Rationale: To check the correct completion of the ADC self-test algorithms.

Implementation hint: Every hardware BIST is activated via a dedicated command sent to the ADC (see the "self-testing" section in the "ADC" chapter of the *MPC5744P Reference Manual* details on implementing these modes).

The SUPPLY SELF-TEST is executed without interleaved conversion (see *MPC5744P Reference Manual* for details).

When using the ADC for an analog input function, additional software tests are required (see Analog inputs).

5.7.6 Cross Triggering Unit (CTU)

The ADC Cross Triggering Unit (CTU) allows automatic generation of ADC conversion requests with minimal CPU intervention. The CTU generates some triggers based on input events.

The trigger can be caused by:

- A pulse
- An interrupt
- An ADC command (or a stream of consecutive commands)
- All of these

If the safety function includes reading of inputs synchronized with events, the system integrator can use the CTU module for this implementation. The required software needed is listed in Synchronize sequentially read inputs.

MPC5744P contains two CTUs, one connected to each PBRIDGE and intended to be used with peripherals connected to the same PBRIDGE. If they are used redundantly along with their connected peripheral modules, no special software test is necessary beyond comparing the overall result of both redundantly used peripherals.

5.7.6.1 Runtime checks

Assumption: [SM_120] The CTU must be properly configured so output triggers are generated within the desired time schedule with respect to the input event(s). [end]

Rationale: To reduce the likelihood of erratic output trigger generation.

A set of ADC commands and pulses generated can be defined for each trigger.

For a detailed description of CTU operation (triggered and sequential mode), its configuration, and use (see the *MPC5744P Reference Manual*).

5.7.6.2 Synchronize sequentially read inputs

Assumption:[SM_121] If the CTU is part of an application safety function, system level functional safety integrity measures for the CTU must be implemented to achieve required integrity. [end]

Rationale: To validate the integrity of the CTU.

Implementation hint: The following mix of hardware mechanisms and software safety integrity measures implemented at the application level provide respective functional safety integrity. Depending on the assignment of inputs to CTU(s) also the existance of two CTUs can be used to fulfill the requirement:

- CTU_HWSWTEST_TRIGGERNUM
- CTU_SWTEST_TRIGGERTIME
- CTU_HWSWTEST_TRIGGEROVERRUN
- CTU_HWSWTEST_ADCCOMMAND (only if the input is an analog signal)
- CTU_SWTEST_ETIMERCOMMAND
- CTU_HW_CFGINTEGRITY

5.7.6.2.1 CTU_HWSWTEST_TRIGGERNUM

If the reload signal occurs before all the triggers are generated, an overrun indication is flagged and the application software may have to handle the error indication.

Rationale: Tests if all the triggers configured within a control period have been generated and serviced.

Implementation hint: The Cross Triggering Unit Error Flag register (CTU_CTUEFR) shows information about the overrun status.

When the CTU detects an error, an interrupt is generated. In the interrupt service routine, the value of the Error Flag Register (CTUEFR) is tested for error condition. If any of the tested bits are valid (= 1, thus an error occurred), appropriate actions may be required.

5.7.6.2.2 CTU_SWTEST_TRIGGERTIME

Application software configures one eTimer channel to capture the time at which each trigger event occurs.

In triggered mode, the time instant of each trigger within one control period is captured and stored in a FIFO. Application software has to check the FIFO values against the expected ones according to CTU configuration.

In sequential mode, an eTimer channel is used to check the correct time of a single trigger with respect to the corresponding event.

Rationale: To check if triggers are generated at the correct time.

Implementation hint: Some eTimer inputs are internally connected to the CTU output. See "Enhanced Motor Control Timer (eTimer)" in the *MPC5744P Reference Manual* for details.

Implementation hint: eTimer capture register implements a two entry FIFO, but in CTU triggered mode up to 8 time values need to be stored. To reduce the likelihood of FIFO overflow condition, eTimer can be configured to trigger a eDMA transfer to move the captured value to specific RAM location.

In sequential mode, an eTimer channel may be needed to check the correct time of a single trigger with respect to the corresponding event.

5.7.6.2.3 CTU_HWSWTEST_TRIGGEROVERRUN

This hardware mechanism checks if a new trigger occurs that requires an action by a subunit that is currently busy. In this case, an overrun interrupt is generated and the application software handles the error condition.

Over-run detection mechanism must be enabled by software during configuration of the CTU.

Rationale: Checks if a new trigger occurs that requires an action by a subunit (for example, ADC command generator) that is currently busy.

Implementation hint: To enable the over-run detection the CTU_CTUIR[IEE] is written with a 1. This interrupt is shared between several sources of error. The application software can determine which particular interrupt is represented by reading the CTU_CTUEFR.

5.7.6.2.4 CTU_HWSWTEST_ADCCOMMAND

The CTU stores in its internal FIFOs both the value provided by each ADC conversion and the channel number. Application software checks the ADC channel number sequence against what is expected for each FIFO. Moreover, invalid commands issued by the CTU are flagged and the corresponding error is handled by the application software (not included in example code).

Rationale: To detect if the incorrect channel has been acquired, or if the incorrect ADC result FIFO is selected.

Implementation hint: To enable detection of invalid commands, the CTU_CTUIR[IEE] flag needs to be asserted. This interrupt is shared between several sources of error. They can be discriminated by reading the CTUEFR register.

This safety integrity function is required only when reading analog signals.

5.7.6.2.5 CTU_SWTEST_ETIMERCOMMAND

Application software configures one channel of eTimer_*n* to count the number of eTimer commands generated within a CTU control period and checks the number against the expected one.

Rationale: To check the correctness of the number of generated commands.

Implementation hint: Some eTimer inputs are internally connected to the CTU output. (See the MPC5744P Reference Manual for details).

5.7.6.2.6 CTU_HW_CFGINTEGRITY

This hardware mechanism ensures the consistency of the CTU configuration at the beginning of each CTU control period.

The configuration registers are all double-buffered. If the configuration is only partial when the control period starts, the previous configuration is used and an error condition is flagged, which is handled by the application software.

Rationale: Ensures the consistency of the CTU configuration.

Implementation hint: The CTU uses a safe reload mechanism. The General Reload Enable (GRE) bit in the Cross Triggering Unit Control Register (CTUCR) has to be used to detect partial or incomplete CTU update. To enable the interrupt in case of error during reload, CTU_CTUIR[IEE] = 1. This interrupt is shared between several sources of error. They can be discriminated by reading the CTUEFR register. Alternatively, repetitive reading of MRS_RE is also possible.

5.7.6.2.7 Other requirements for the CTU

Assumption: [SM_123] If the CTU is used to read an analog signal through the ADC, the software must check the Invalid Command Error flag (CTU_CTUEFR[ICE]) after programming the ADC command lists. [end] Another option would be to use the two redundant CTUs (on different PBRIDGEs) and compare results.

Rationale: To check the presence of invalid commands.

5.7.7 Wake-Up Unit (WKPU) / External NMI

Assumption under certain conditions:[SM_126] If external NMI and Wake-up are used as a safety mechanism, especially if waking up within a certain timespan or at all is considered safety-relevant, it is required to implement corresponding system level measures to detect latent faults in the WKPU. [end]

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

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.

5.7.8 Glitch filter

An analog glitch filter is implemented on the reset signal of the MPC5744P. A selectable (WKPU_NCR[NFE0]) 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 MPC5744P 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 MPC5744P. The following methods for deriving the base failure rates of the MPC5744P 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 MPC5744P 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 MPC5744P enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request.

6.2.1 Module classification

For calculating the safety metrics for ISO 26262 (Single-Point Failure Metric (SPFM), Latent Failure Metric (LFM) and Probabilistic Metric for random Hardware Failures (PMHF)) and for IEC 61508 (Safe Failure Fraction (SFF) and $ß_{IC}$ factor) the modules of the MPC5744P 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 MPC5744P can be found in the attached "MPC5744P 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 MPC5744P 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 MPC5744P and is capable of detecting this CCF, and will switch the system to a Safe state$_{system}$ within the FTTI.

The MPC5744P device satisfies the standard AECQ100 for latch-up immunity.

7.1.2.2 Environmental conditions

7.1.2.2.1 Temperature

The MPC5744P was designed to work within a maximum operational temperature profile (see the *MPC5744P Data Sheet*). To cover temperature-related dependent failures, two temperature sensors for supervision are implemented as described in section "Temperature Sensors (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 (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 MPC5744P or by detecting such interference. For example, there must be assurance that the system is not debugged (or unintentionally placed in debug mode), or placed in any other special mode different from normal application execution mode (for example, test mode). In addition, an FCCU failure indication is generated if:

- A self-test sequence of the STCU is unintentionally executed during normal operation of the device.
- 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 *MPC5744P 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 *MPC5744P Data Sheet*) together with the ball number information of the packages as seen in the *MPC5744P 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 MPC5744P 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 MPC5744P 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 MPC5744P 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 MPC5744P. As soon as there are no correct communications, the external watchdog function switches the system to Safe $state_{system}$. Thus, either the MPC5744P or external watchdog function can transition the system to Safe $state_{system}$. The external watchdog function is required to be sufficiently independent of the MPC5744P (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 β_{IC} considerations

During the development of the MPC5744P, the susceptability of the MCU to dependent failures is evaluated by ensuring sufficient independence between on-chip functions and their respective safety mechanisms.

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

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

Chapter 8
Additional Information

8.1 Testing All-X in RAM

As mentioned in section End-to-end ECC (e2eECC), All-0 or All-1 content will be an uncorrectable error only at some addresses in RAMs where address is included in the ECC calculation. This section contains a program which provides these adresses and can thus be used to either determine an address to periorically read or check whether addresses which are periodically read by an application show this desired behaviour.

8.1.1 Candidate address for testing All-X issue

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

```
#--- start Perl script ---:
eval 'exec perl -w -S $0 ${1+"$@"}'
  if 0;
use strict;
my $base = hex($ARGV[0]);
my $num_to_find = ($#ARGV > 0) ? $ARGV[1] : 1;
my $all0_found = 0;
my $all1_found = 0;
my $guesses = 0;
my $addr = $base;
my $ecc;
my $bit_count;
printf "RAM base address = 0x%08x\n", $base;
printf "  All 0s - Addresses with two bits set in the address ECC contribution:\n";'
while(($guesses < 131072) && ($all0_found < $num_to_find)) {
      $ecc = get_ecc($addr, 0, 0);
      $bit_count = count_ones($ecc);
      if($bit_count == 2) {
      $all0_found++;
      printf "    (%d) addr = 0x%08x, addr_ecc = 0x%02x\n", $all0_found, $addr, $ecc;
}
$addr += 8;
$guesses++;
}
printf "\n  All 1s - Addresses with two bits cleared in the address ECC contribution:\n";
$addr = $base;
while(($guesses < 131072) && ($all1_found < $num_to_find)) {
      $ecc = get_ecc($addr, 0xffffffff, 0xffffffff);
```

```
      $bit_count = count_zeroes($ecc);
      if($bit_count == 2) {
          $all1_found++;
          printf "    (%d) addr = 0x%08x, addr_ecc = 0x%02x\n", $all1_found, $addr, $ecc;
      }
      $addr += 8;
      $guesses++;
}
sub count_ones {
      my $string = sprintf("%08b", shift);
      my $count = 0;
      my $i;
      for($i=0; $i<8; $i++) {
          if(substr($string, $i, 1) eq "1") {
              $count++;
          }
      }
      return($count);
}
sub count_zeroes {
      my $string = sprintf("%08b", shift);
      my $count = 0;
      my $i;
      for($i=0; $i<8; $i++) {
          if(substr($string, $i, 1) eq "0") {
              $count++;
          }
      }
      return($count);
}
sub get_ecc {
my $addr = shift;
my $data_be0 = shift;
my $data_be1 = shift;

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

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

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

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

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



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

my $addr_ecc_tcm
    = (0x1f & $addrx8[31])
    ^ (0xf4 & $addrx8[30])
    ^ (0x3b & $addrx8[29])
    ^ (0xe3 & $addrx8[28])
    ^ (0x5d & $addrx8[27])
    ^ (0xda & $addrx8[26])
    ^ (0x6e & $addrx8[25])
    ^ (0xb5 & $addrx8[24])
    ^ (0x8f & $addrx8[23])
    ^ (0xd6 & $addrx8[22])
    ^ (0x79 & $addrx8[21])
    ^ (0xba & $addrx8[20])
    ^ (0x9b & $addrx8[19])
    ^ (0xe5 & $addrx8[18])
    ^ (0x57 & $addrx8[17])
    ^ (0xec & $addrx8[16])

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

```
    ^ (0xf2 & $addrx8[4])
    ^ (0x2f & $addrx8[3])
my $data_ecc
    = (0xb0 & $data_lex8[63])
    ^ (0x23 & $data_lex8[62])
    ^ (0x70 & $data_lex8[61])
    ^ (0x62 & $data_lex8[60])
    ^ (0x85 & $data_lex8[59])
    ^ (0x13 & $data_lex8[58])
    ^ (0x45 & $data_lex8[57])
    ^ (0x52 & $data_lex8[56])

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

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

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

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

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

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


    ^ (0x8c & $data_lex8[7])
    ^ (0x4a & $data_lex8[6])
    ^ (0x4c & $data_lex8[5])
```

```
    ^ (0x15 & $data_lex8[4])
    ^ (0x83 & $data_lex8[3])
    ^ (0x9e & $data_lex8[2])
    ^ (0x43 & $data_lex8[1])
    ^ (0xc1 & $data_lex8[0])

  my $ecc       = $data_ecc ^ $addr_ecc;
  my $ecc_tcm   = $data_ecc ^ $addr_ecc ^ $addr_ecc_tcm ^ 0x55;
  my $ecc_flash = $data_ecc ^ 0xff;
  return($ecc);
}
##printf "addr        = 0x%08x\n", $addr;
##printf "data_be     = 0x%08x_%08x\n", $data_be0, $data_be1;
##printf "addr_ecc    = 0x%02x\n", $addr_ecc;
##printf "data_ecc    = 0x%02x\n", $data_ecc;
##printf "ecc         = 0x%02x\n", $ecc;
##printf "ecc_tcm     = 0x%02x\n", $ecc_tcm;
##printf "ecc_tcm_fix = 0x%02x\n", $ecc_tcm_fix;
##printf "ecc_flash   = 0x%02x\n", $ecc_flash;
#----- end perl script -----
```

This script finds the first N addresses with 2 or 6 bits set and 2 or 6 bits cleared in the address ECC contribution. Usage is as follows:

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

Example:

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

RAM base address = 40000000h

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

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

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

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

RAM base address = 40000000h

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

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

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

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

8.1.2 ECC checkbit/syndrome coding scheme

The e2eECC scheme implements a single-error correction, double-error detection (SECDED) code using the so-called Hsiao odd-weight column criteria. These codes are named for M.Y. Hsiao, an IBM researcher who published extensively in the early 1970s on SECDED codes better suited for implementation in protecting (mainframe) computer memories than traditional Hamming codes.

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

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

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

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

Table 8-1. e2eECC basic H-matrix definition

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

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

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

Figure 8-1. C Language encode ECC function description

```
encodeEcc (addr, data_a2_is_zero, data_a2_is_one)
    unsigned int        addr;                    /* 32-bit byte address */
```

```
    unsigned int        data_a2_is_zero;        /* 32-bit data lower, a[2]=0 */
    unsigned int        data_a2_is_one;         /* 32-bit data upper, a[2]=1 */

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

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

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

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

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

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

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

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

        ^ (((data_a2_is_zero >> 15) & 1) ? 0x54 : 0x0)      /* data[47] */
        ^ (((data_a2_is_zero >> 14) & 1) ? 0xa7 : 0x0)      /* data[46] */
        ^ (((data_a2_is_zero >> 13) & 1) ? 0xa8 : 0x0)      /* data[45] */
        ^ (((data_a2_is_zero >> 12) & 1) ? 0x92 : 0x0)      /* data[44] */
        ^ (((data_a2_is_zero >> 11) & 1) ? 0xc8 : 0x0)      /* data[43] */
        ^ (((data_a2_is_zero >> 10) & 1) ? 0x07 : 0x0)      /* data[42] */
        ^ (((data_a2_is_zero >>  9) & 1) ? 0x34 : 0x0)      /* data[41] */
        ^ (((data_a2_is_zero >>  8) & 1) ? 0x32 : 0x0)      /* data[40] */
```

```
        ^ (((data_a2_is_zero >>  7) & 1) ? 0x68 : 0x0)       /* data[39] */
        ^ (((data_a2_is_zero >>  6) & 1) ? 0x89 : 0x0)       /* data[38] */
        ^ (((data_a2_is_zero >>  5) & 1) ? 0x98 : 0x0)       /* data[37] */
        ^ (((data_a2_is_zero >>  4) & 1) ? 0x49 : 0x0)       /* data[36] */
        ^ (((data_a2_is_zero >>  3) & 1) ? 0x61 : 0x0)       /* data[35] */
        ^ (((data_a2_is_zero >>  2) & 1) ? 0x86 : 0x0)       /* data[34] */
        ^ (((data_a2_is_zero >>  1) & 1) ? 0x91 : 0x0)       /* data[33] */
        ^  ((data_a2_is_zero        & 1) ? 0x46 : 0x0)       /* data[32] */

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

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

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

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

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

    return(ecc);
}
```

On a memory read operation, the e2eECC logic performs the same type of optional adjustment on the read checkbits.

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

Chapter 9
Acronyms and Abbreviations

9.1 Acronyms and abbreviations

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

Table 9-1. Acronyms and abbreviations

Terms	Meanings
BAM	Boot Assist Module
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
MEMU	Memory Error Management Module
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

Table continues on the next page...

Table 9-1. Acronyms and abbreviations (continued)

Terms	Meanings
SPF	Single-Point Fault
SPFM	Single-Point Faults Metric
TED	Triple-Error Detection

Appendix A
Release Notes for MPC5744P

A.1 MPC5744P Safety Manual general changes

• Editorial updates and improvements throughout. • Removed the "MPC5744P Physcial Pin Displacement on Internal Die" spreadsheet attachment; see the "Signal Description" chapter in the MPC5744P Reference Manual instead.

A.2 MPC5744P Safety Manual chapter changes

A.2.1 Changes: Preface

• In Safety manual assumptions and Related documentation, changed "dynamic FMEDA" to "FMEDA".
• 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".

A.2.2 Changes: MCU Safety Context

• In MCU fault indication time, changed "The three mechanisms" to "The mechanisms".
• In Correct operation :   • Changed "Software Execution Function" to "MCU Safety Function".

• In the Faults section:
  • Editorial changes to the "Latent Fault" bullet for better clarity.
• In the Latent-fault tolerant time interval for latent faults section:
  • Added Assumption SM_212 and Rationale.

• In the Safety integrity level section: • Changed "ISO26262" to "ISO 26262", and "IEC61508" to "IEC 61508".
• In the MCU safety functions section: • Added the sentence/paragraph "Please see the Module classification section for more details" at the end of the section.

• In Failure types :   • Updated the explanation of random hardware fault reduction and detection for improved clarity.
• In the MCU safety functions section: • Changed the "Debug Functions" bullet to "Not Safety Related functions: It is assumed that some functions are Not Safety Related (e.g. debug)."
• Cleaned up Figure 2-1. • In the Faults and failures section: • Modified the hierarchy of this section - no content changes.

A.2.3 Changes: MCU Safety Concept

• In the Memory Error Management Unit (MEMU) section:
  • Changed "system RAMs in this context are those RAMs with "System RAM" memory classification" to "a system RAM in this context is any RAM that is CPU accessible".
• In the Interface to ECC units section:
  • In the "configuration" bullet, changed "vital memory (SRAM, local memory, and system RAM of Safety Core)" to "safety-related system RAM".

• In Power, changed "for voltages that are out of these ranges" to "for values that are out of these ranges".
• In the Clock section: • Changed "SW" to "Software" in the NOTE.

• In the End-to-End protection on data path section:
  • Changed "Safety Core" to "core".
  • Changed "replicated cores" to "masters".
• In the Failure handling section:
  • Under "Available failure reactions", changed the "Resetting the chip" bullet to "Resetting the MCU".
• In the Operational interference protection section:
  • Changed "Safety Core" to "core".

• In the ECC for storage section:
  • Changed "... these same memories include dedicated measures..." to "... some of these memories include dedicated measures..."
  • Changed "FlexCAN" to "CAN"

• In Disabling of communication controllers :
  • In the last paragraph, changed "(for example, FlexCAN, ENET)" to "(for example, CAN, ENET, FlexRay)".
• In Failure handling :
  • Under "Available failure reactions", in the bullet "Disabling the transmission capabilities... ", added "FlexRay" in the example list.
  • Changed "FlexCAN" to "CAN".

A.2.4 Changes: Hardware Requirements

• In PowerSBC :
  • Changed the supply pin names in the figure to be generic.
  • Added a note referring the reader to the Data Sheet for the correct supply pin names.
• In PWM output monitor :
  • Removed the module examples for simplicity.

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

- In Assumption [SM_088], changed "the maximum survivable voltage of the technology" to "the absolute maximum rating of the device (as documented in the Data Sheet in section "Absolute maximum ratings")". - Removed the recommendation to disable the system when an overvoltage occurs.

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

- In the Hardware requirements on system level section:
  - Simplified the figure.
  - Removed the Note.

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

- In the Power Supply Monitor (PSM) section:
  - Changed the word "powerless" to "unpowered".
- In the PWM output monitor section:
  - In the list of features to be managed by system level measures, removed the example at the end of the "Open GPIO" bullet.

- In the Error Out Monitor (ERRM) section: - Changed "FCCU_F[0], and optionally FCCU_F[1]" to "FCCU_F[0] and/or FCCU_F[1]".
- In External Watchdog (EXWD) :   - In the bullet "Periodic message frames", added "Flexray".

A.2.5 Changes: Software Requirements

- In the Hardware element section: - Corrected cross reference to non-existent "Figure 14" to be Figure 5-5.
- In the Clock Monitor Unit (CMU) Clock Monitor Unit (CMU) section: - Changed occurrences of "All five CMUs" to "The CMUs".

- In the Software watchdog timer section:
  - Changed occurrences of "FTTI/PST" to "FTTI".
  - Removed Assumption SM_203 (redundant to SM_067).
- In the Periodic Interrupt Timer (PIT) Runtime checks section:
  - Removed Assumption SM_108 (redundant to SM_070).
- In the Cyclic Redundancy Checker Unit (CRC) Runtime checks section:
  - Removed Assumption SM_071 (redundant to SM_140) and the associated note.
  - In the Implementation details subsection:
    - Changed all tests from MODULE0/1/2/etc to MODULE*n*.
- In the Temperature Sensors (TSENS) section:
  - In the "Implementation hint":
    - Removed the list of temperature thresholds.
    - Referred the reader to the Temperature Sensor chapter of the Reference Manual for further information.
- In the Cross Triggering Unit (CTU) section:
  - Removed the lists of example events and example peripherals.
  - In the CTU_SWTEST_ETIMERCOMMAND section:
    - Changed specific instances of eTimer_0/eTimer_1/eTimer_2 to eTimer_*n*.
- In Dual PLL Digital Interface (PLLDIG) :

• Changed "The auxiliary clock from the non-FMPLL is instead distributed to those peripherals that require precise timing (for example, eTimer, FlexPWM), its clock is not modulated" to "The auxiliary clock from the non-FMPLL is not modulated, and is distributed to those peripherals that require precise timing".

• Throughout the External Oscillator (XOSC) section:
  • Changed "FlexCAN" to "CAN"
• In the Non-replicated eDMA transfers section:
  • Removed reference to FlexCAN/CAN
• Throughout the Communications section:
  • Changed "FlexCAN" to "CAN"
• In the Analog to Digital Converter (ADC) section:
  • Changed "Analog-to-Digital Converter (ADC)" to "Successive Approximation Register (SAR) Analog-to-Digital Converter (ADC)"
  • Removed extraneous text, "on independently"

• In the Temperature Sensors (TSENS) section:
  • In the Implementation hint:
    • Changed "When the temperature threshold detection feature is enabled..." to "When the temperature sensor is enabled...".
    • In the Note, changed "Temperature thresholds..." to "The over and under temperature thresholds...".
    • Changed "The sensor output..." to "The sensor analog output...".

• In PLLDIG Initial checks and configurations :   • In the first "Implementation hint", changed "externally provided clock reference" to "internally provided clock reference".
• In the 1.25 V supply supervision section: • In the Implementation hint, removed the content related to external ballast transistors.

• In the Memory Error Management Unit (MEMU) section:
  • Merged the first paragraph of the MEMU "Runtime checks" subsection into this one, and removed the remainder of that subsection.
• In the ADC Initial checks and configurations section:
  • Changed the end of the Assumption from "safety integrity measures must be implemented after reset (external reset or destructive reset) before starting the respective safety function to ensure ADC integrity" to "safety integrity measures must be implemented once per L-FTTI".

• In the Power Management Controller (PMC) section:
  • In the discussion of LVD/HVD in the first paragaraph, changed "cause a potential failure" to "could cause a potential failure".
• In the PLLDIG Initial checks and configurations section:
  • Added the comma to "functional safety integrity measure, respective functional..." for clarity.

• In the MPC5744P configuration section: • Removed "Most of the off-platform peripherals have their own REG_PROT."
• In System Memory Protection Unit (SMPU) :   • Changed "Each XBAR master (Core, DMA, SIPI)..." to "Each XBAR master (Core, DMA, FlexRay, Ethernet, SIPI)..."
• In Test mode :   • Removed the recommendation "Use system level software measure to disable test mode."
• In Enhanced Direct Memory Access (eDMA), changed “As eDMA is a replicated module, no software action” to “As eDMA is a module in lockstep, no software action”. • In SMPU Initial checks and configurations, removed the parenthetical example after "If non-replicated bus masters". • In FCCU Initial checks and configurations, added a sentence that begins with "This chip’s reference manual includes either a table or an attached file" to the implementation hint.

• In Power Management Controller (PMC) :
  • In the first paragraph, changed "and high voltage detectors (HVD)" to "and a high voltage detector (HVD)".
• In 1.25 V supply supervision :
  • Changed the end of the last sentence above the figure from "a destructive reset is triggered" to "a destructive reset, a functional reset, or an interrupt is triggered."
  • In the Implementation hint, removed the sentence, "Alternatively, an external 1.25 V HVD..."
• In Debug mode :
  • Changed "FlexCAN" to "CAN".

• In XBAR Runtime checks :
  • Changed "FTTI/PST" to "FTTI".
• In Temperature Sensors (TSENS) :
  • Changed "Two temperature sensors monitor the substrate temperature to detect over-temperature conditions before..." to "Two temperature sensors monitor the substrate temperature to detect over-temperature conditions and under-temperature conditions before..."
• In ADC Initial checks and configurations :
  • Changed the first Implementation hint from "After reset (external reset or destructive reset), but before executing any safety function, the following hardware BISTs..." to "Once per L-FTTI, the following hardware BISTs..."
• In the Digital inputs section, in Double read PWM inputs :
  • In subsection "ETIMER1_SWTEST_CMP", removed the short list of eTimer instances.
• In the Digital inputs section, in Synchronize sequential read input :
  • Removed the lists of example inputs/events for which the CTU can be used.
• In Digital outputs :
  • Removed the device number from the figure.

• Made editorial changes.
• In ADC Initial checks and configurations, rewrote the implementation hint for the assumption under certain conditions that begins with "Before running the ADC hardware self-test".

A.2.6 Changes: Failure Rates and FMEDA

• In the FMEDA section: • Changed "Dynamic FMEDA" to "FMEDA".
• In the Module classification section, changed the term "Software Execution Function" to "MCU Safety Functions".

A.2.7 Changes: Dependent Failures

• Throughout the chapter:
  • Changed "CMF" to "dependent failure".
  • Changed "common mode failure" to "dependent failure".
• In Causes of dependent failures :
  • Changed bullet "physical defects that are able to influence an element and its redundant element" to "dependent failures that are able to influence an on-chip function and its respective safety mechanisms".
• In Nonapplication control signals :
  • In the first paragraph:
    • Changed the phrase "functional safety-relevant and thus have no functional safety mechanism included" to "safety-related".
    • Changed the phrase "violate the functional safety goal" to "lead to a safety-related failure".
    • Changed the phrase "functional safety-relevant" to "safety-related".
  • In the list of FCCU failure indication conditions, removed the bullet "The device leaves LSM".
• In External timeout function :
  • Updated the NOTE.
• In β_{IC} considerations :
  • Changed "Dynamic FMEDA" to "FMEDA".

• In Modules sharing PBRIDGE :   • Removed the example, as it is unnecessary.
• In I/O pin/ball configuration :   • Changed references to the "Physcial Pin Displacement on Internal Die" attachment to "The Signal Description chapter in the MPC5744P Reference Manual".
• In Dependent failure avoidance on system level :

• Removed the Recommendation as it was redundant. • In External timeout function : • Removed the sentence that immediately followed the note, as it was redundant.
• In Physical isolation : • In the second paragraph, changed the word "fatal" to "catastrophic".
• In External timeout function : • In the last line of the last paragraph, changed "communication (such as via CAN)" to "communication (such as via CAN or FlexRay)".

A.2.8 Changes: Additional Information

• In Testing All-X in RAM : • Changed "E2E ECC" to "e2eECC" throughout the section.
• In the ECC checkbit/syndrome coding scheme section: • Added the following footnote to the "Data Bit" heading (applies to both the "Data Bit" and "Address Bit" headings) in Table 8-1 : "Bit numbering is AHB convention, bit 0 is LSB. D[7:0] corresponds to byte at address 0. D[63:56] corresponds to byte at address 7."

A.2.9 Changes: Acronyms and Abbreviations

• Updated the list of acronyms.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MPC5744PSM
Revision 4, 02/2018