Safety Manual for MPC5777C

Devices Supported: MPC5777C

Document Number: MPC5777CSM
Rev. 2.1, 02/2017

Contents

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

Chapter 1
Preface

Chapter 2
MCU safety context

Chapter 3
Functional safety concept

Chapter 4
Hardware requirements

Chapter 5
Software requirements

Chapter 6
Failure rates and FMEDA

Chapter 7
Dependent failures

Chapter 8
Additional information

Section number Title Page

Chapter 1
Preface

1.1 Overview

This document discusses requirements for the integration and use of the MPC5777C 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 MPC5777C, 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 MPC5777C is developed according to ISO 26262 and has an integrated safety concept.

1.2 Safety manual assumptions

During the development of the MPC5777C, assumptions were made on the system level safety requirements with regards to the MCU. During the system level development, the safety system developer is required to establish the validity of the MCU assumptions in the context of the specific safety-related system. To enable this, all relevant MCU assumptions are published in the Safety Manual and can be identified as follows:

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

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

Example: **Assumption under certain conditions:** If a high impedance state on an output is not safe, pull-up or pull-down resistors shall be added to safety-critical outputs. The need for this will be application dependent for the unpowered or reset condition (tristated I/O) of the MPC5777C.

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 MPC5777C 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 MPC5777C. 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 MPC5777C 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 MPC5777C is considered a Safety Element out of Context (SEooC) development, as described in *ISO 26262-10.9 Safety element out of context* and more specifically detailed in *ISO 26262-10.9.2.3 Development of a hardware component as a safety element out of context* and *ISO 26262-10 Annex A ISO 26262 and microcontrollers*.

1.5 Related documentation

The MPC5777C 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 MPC5777C into safety-related systems, the following documentation will be available:

- Reference Manual (MPC5777CRM) - Describes the MPC5777C functionality
- Data Sheet (MPC5777CDS) - Describes the MPC5777C operating conditions
- Safety Manual (MPC5777CSM) - Describes the MPC5777C safety concept and possible safety mechanisms (integrated in MPC5777C, system level hardware or system level software), as well as measures to reduce dependent failures
- FMEDA - Inductive analysis enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request. The MPC5777C 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 MPC5777C, the following information should be considered:

- The MPC5777C is handled in accordance with JEDEC standards J-STD-020 and J-STD-033.
- The operating conditions given in the MPC5777C Data Sheet.
- If applicable, any published MPC5777C errata.

- The recommended production conditions given in the MPC5777C quality agreement.
- The safety system developer is required to report all field failures of the MPC5777C to NXP.

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

Chapter 2
MCU safety context

2.1 Safety integrity level

The MPC5777C 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 MPC5777C 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 MPC5777C is considered a Safety Element out of Context (SEooC) development.

The MPC5777C 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).

2.2 Safety function

2.2.1 MCU safety functions

Given the application independent nature of the MPC5777C, no specific safety function can be specified. Therefore, during the SEooC development of the MPC5777C, 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 MPC5777C flash memory, buffer these within instruction cache, execute instructions, read data from the MPC5777C System SRAM or flash memory, buffer these in data cache, process data and write result data into MPC5777C System

SRAM. **Functional safety of the Software Execution Function is primarily achieved by safety mechanisms integrated on the MPC5777C**.

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

- **Input / Output Functions (Application dependent)**: Input / Output functions of the MPC5777C 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.

2.2.2 Correct operation

Correct operation of the MPC5777C 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.

2.3 Safe states

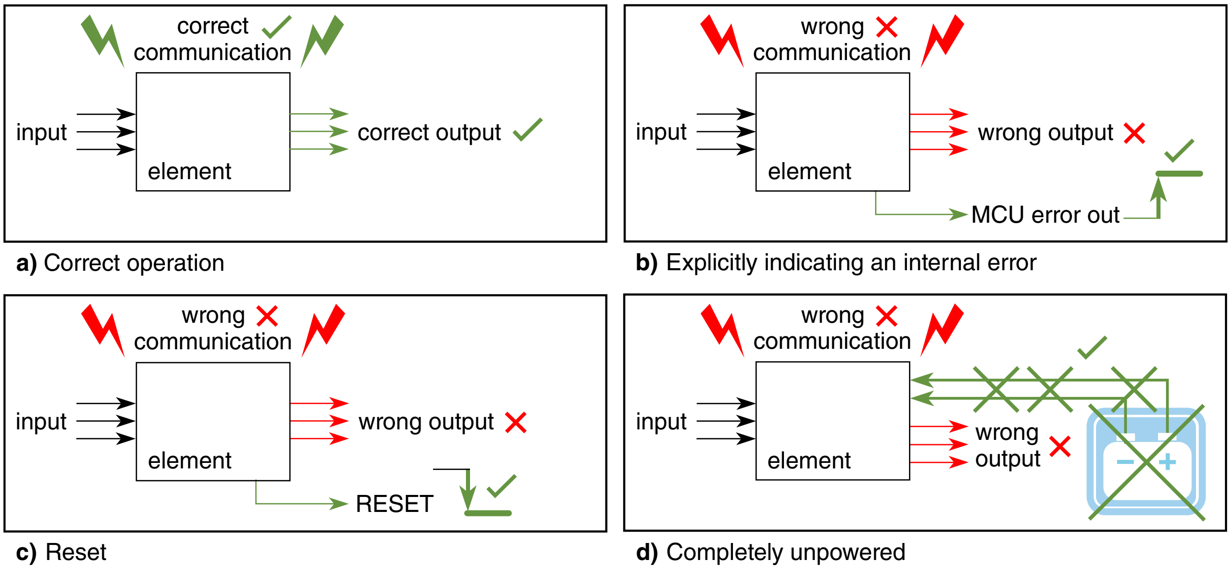
A safe state of the system is named Safe state$_{\text{system}}$, whereas a safe state of the MPC5777C is named Safe state$_{\text{MCU}}$. A Safe state$_{\text{system}}$ is an operating mode without an unreasonable probability of occurrence of physical injury or damage to the health of any persons. A Safe state$_{\text{system}}$ may be the intended operating mode or a mode where the system has been disabled.

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

2.3.1 MCU Safe state

The safe states (Safe state$_{MCU}$) of the MPC5777C are:

- Operating correctly (see Figure 2-1 and section "Correct operation")
- Explicitly indicating an internal error (indication on ERROR*n*, Figure 2-1)
- In reset (see Figure 2-1)
- Completely unpowered (see Figure 2-1)



a) Correct operation

b) Explicitly indicating an internal error

c) Reset

d) Completely unpowered

Figure 2-1. Safe state$_{MCU}$ of MPC5777C

2.3.2 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 ERROR0 or ERROR1). [end]

Implementation hint: If the MPC5777C signals an internal failure via its error out signals (ERROR*n*), the surrounding subsystem shall no longer use the MPC5777C 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 MPC5777C. Depending on the configuration, the system may disable, or reset, the MPC5777C 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$_{\text{system}}$ when the MCU is unpowered. [end]

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

2.3.3 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]

2.4 Faults and failures

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.

2.4.1 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 MPC5777C.
- **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.

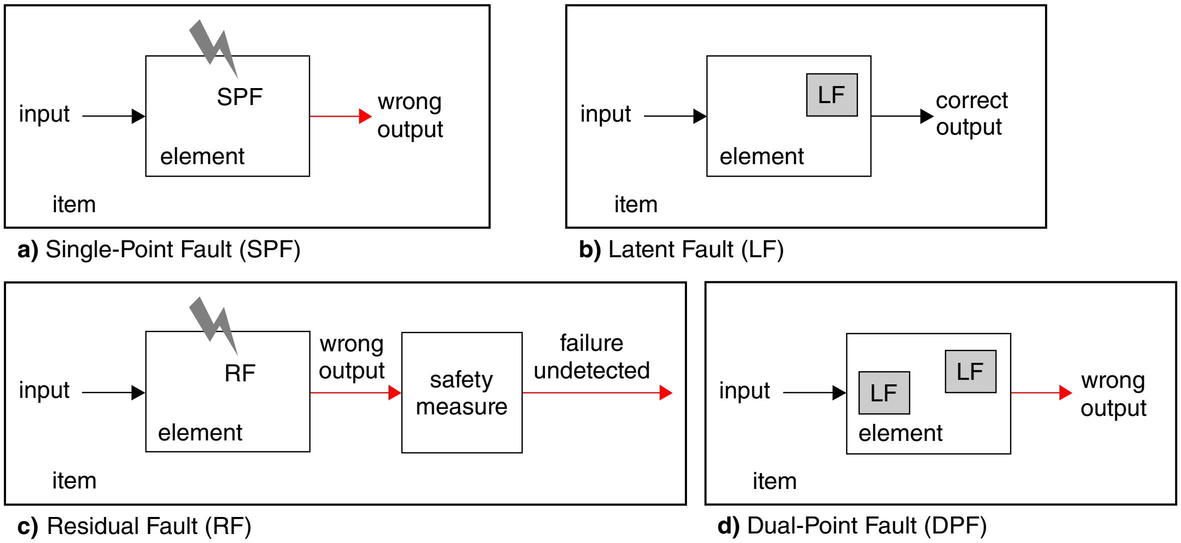


Figure 2-2. Faults

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}.

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

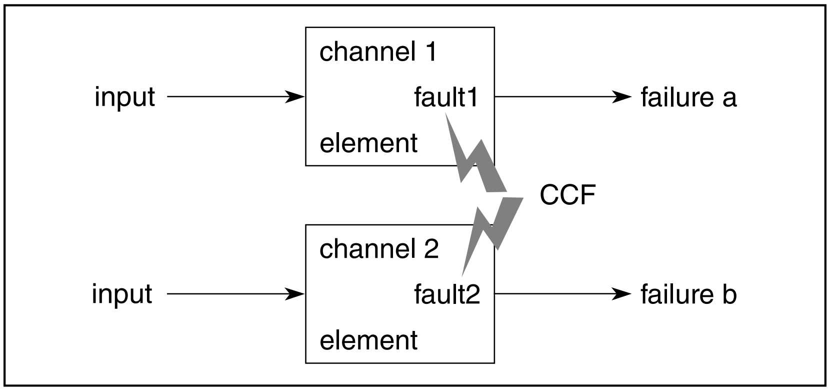


Figure 2-3. Common Cause Failures

- **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.

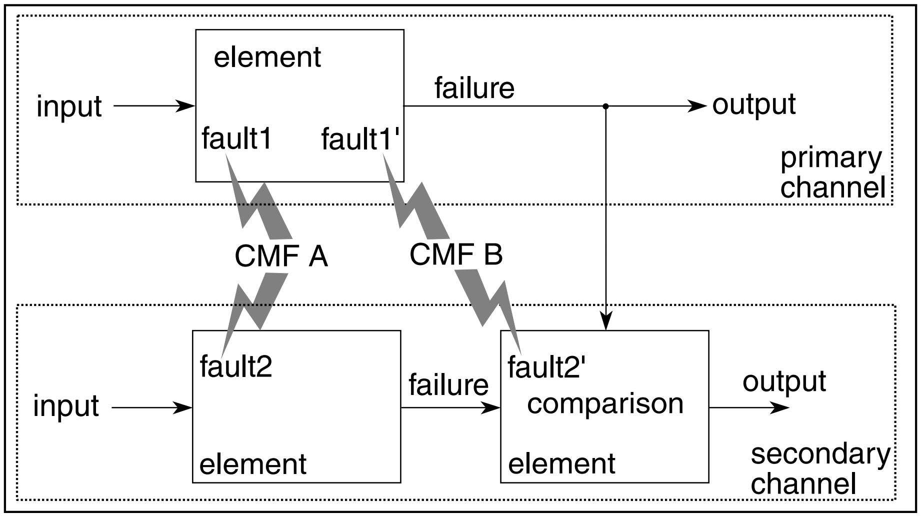


Figure 2-4. Common Mode failures

- **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).

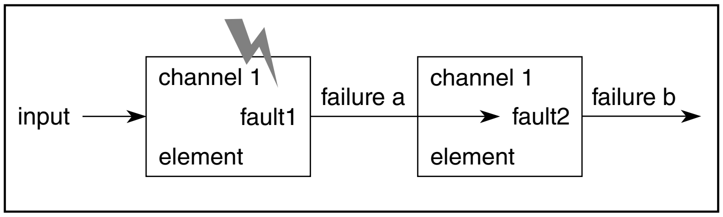


Figure 2-5. Cascading failures

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

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

Figure 2-6 shows the FTTI for a system:

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

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

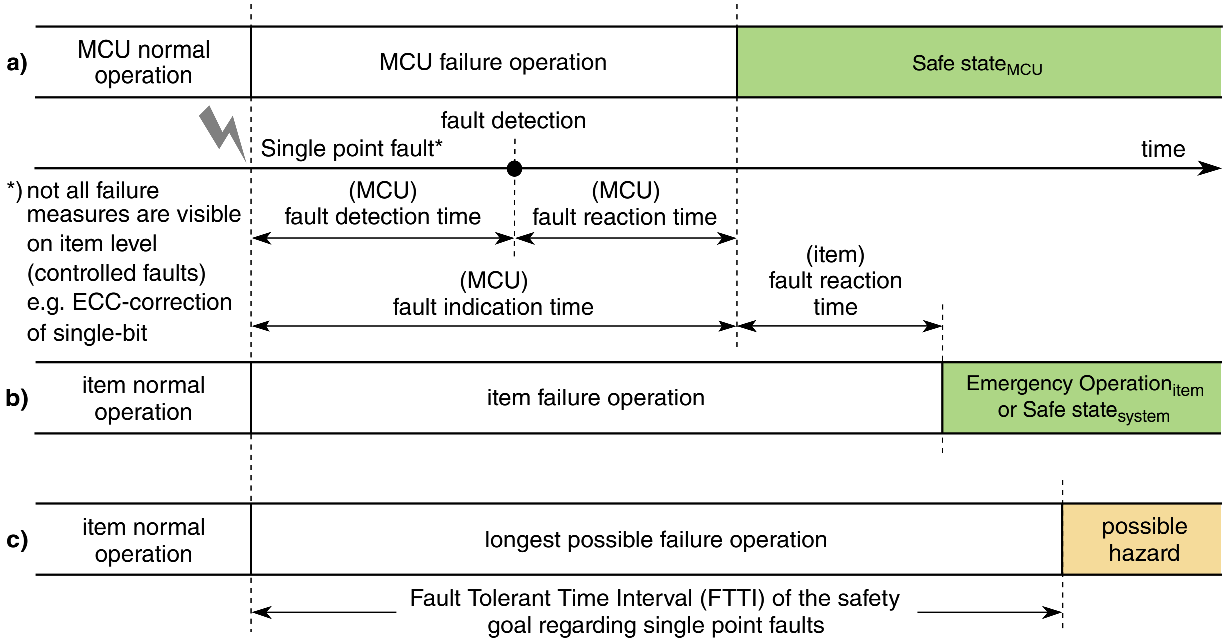

a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
fault detection
Single point fault*
time
*) not all failure measures are visible on item level (controlled faults) e.g. ECC-correction of single-bit
(MCU) fault detection time
(MCU) fault reaction time
(MCU) fault indication time
(item)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

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

Fault indication time is the time from the occurrence of a fault to when the MPC5777C is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the MPC5777C to a high impedance state, or by assertion of reset).

2.5.1 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 MPC5777C. 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 (ERROR*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 MPC5777C fault indication time and system fault reaction time should be less than the FTTI of the functional safety goal.

2.6 Latent-fault tolerant time interval for latent faults

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

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

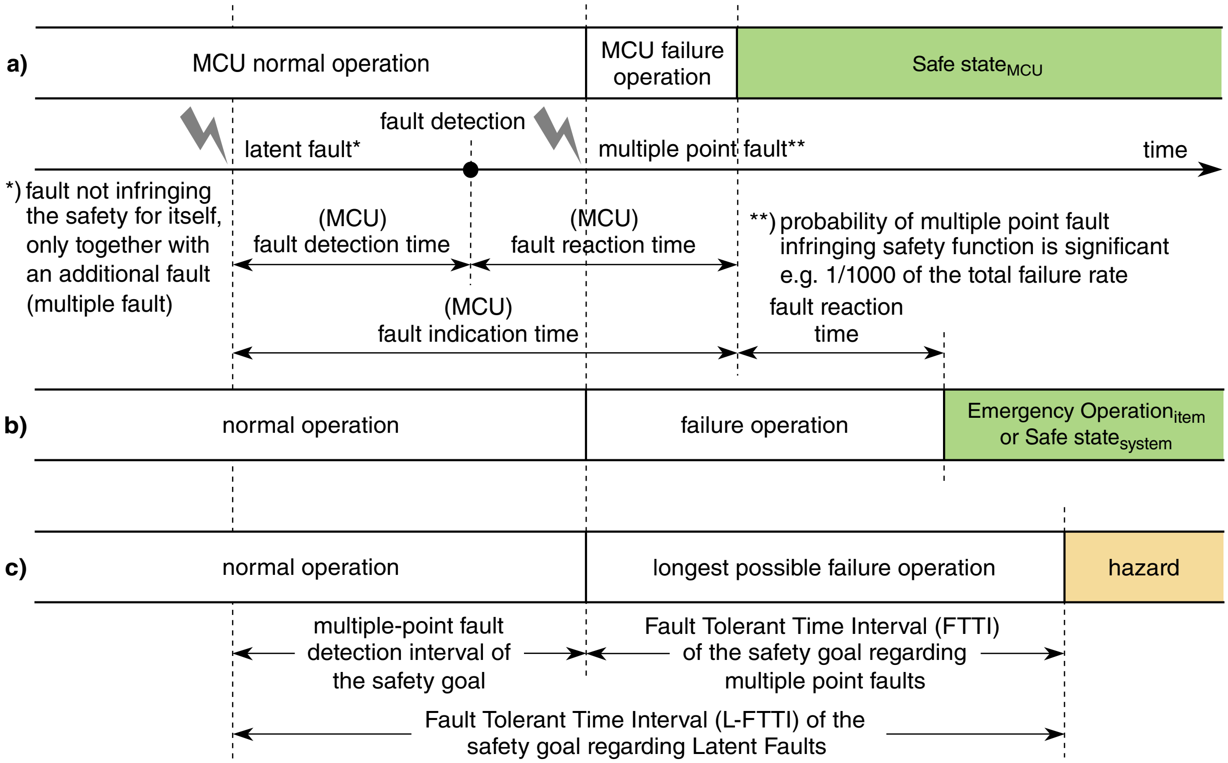

a)
MCU normal operation
MCU failure operation
Safe state$_{MCU}$
latent fault*
fault detection
multiple point fault**
time
*) fault not infringing the safety for itself, only together with an additional fault (multiple fault)
(MCU) fault detection time
(MCU) fault reaction time
**) probability of multiple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

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

Latent fault indication time is the time it takes from the occurrence of a multiple-point failure to when the indication of that failure is driven on ERROR*n*, forcing the outputs of the MPC5777C 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.

2.6.1 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.

2.7 MCU failure indication

2.7.1 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 MPC5777C 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 MPC5777C.

2.7.2 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 MPC5777C. The FCCU provides two error output signals (ERROR0 and ERROR1) 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 ERROR*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 ERROR*n* outputs will transition to the operational state only by software request.

At least one of the ERROR*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, ERROR0 = 0 and ERROR1 = 1 and vice-versa), the application software can configure that ERROR*n* signal that will be high to indicate the operational state (see Error Out Monitor (ERRM) for details on requirements for connecting ERROR*n* to external devices).

Chapter 3
Functional safety concept

3.1 General functional safety concept

Figure 3-1 shows a block diagram of the MPC5777C.

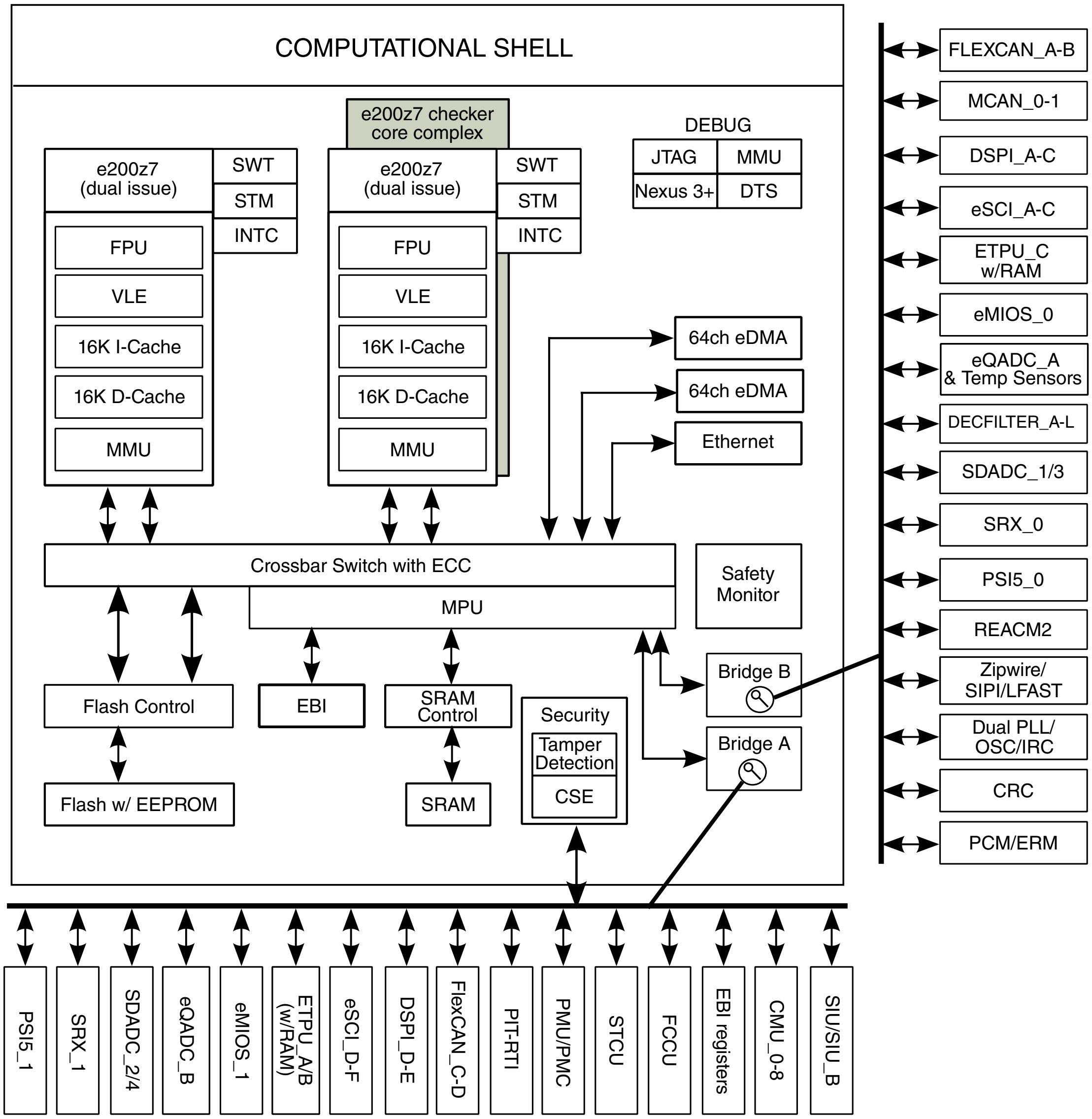


Figure 3-1. MPC5777C block diagram

Functional Safety integrity measures are:

- Replication of IP: A dual core architecture reduces the need for component duplication at the system level, and lowers overall system complexity.
- Replication of processing elements are as follows:
- For the dual lockstep z7 cores (Core 1 and Checker) and cache controllers, functional safety is ensured by a lockstep approach. Any deviation in the output of the two lockstep z7 cores is detected by hardware and signaled as a possible failure.
- Error correction or detection to reduce the effect of faults in the following integrated volatile and non-volatile memories:
  - Flash memory

  - SRAM
  - FlexCAN
  - MCAN
  - ENET
  - Cache and cache tags
  - eDMA Transfer Control Descriptor (TCD) RAM
  - eTPU
- The generation and distribution of clock and power are supervised by dedicated monitors.
- Built-in self tests (for example, MBIST and LBIST) are implemented in hardware to detect in general latent failures and therefore reducing the risk of coincident failures (multiple-point faults).
- The FCCU is responsible for collecting and reacting to failure notifications.
- CMF are dealt with by a set of measures for both control and reduction, spanning system level approaches (such as temperature and non-functional signal monitoring), physical separation, and diversity.
- The functional safety of the periphery is ensured by application level (system level) measures (such as connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on). For this, the chip ensures that redundant use of peripherals is protected against CMF.
- Usage of internal (and external) watchdogs or timeout measures.
- Dedicated mechanisms are suggested to check the functionality of error reaction paths (such as by application controlled fault injection).

The MPC5777C safety core operates in delayed lockstep mode (LSM) to allow the highest safety level to be reached. The checker core will receive all inputs delayed by two clock cycles. Outputs of the checker core will be compared with outputs of the master core. Any differences will be flagged as an error and processed by the FCCU.

LSM is enabled or disabled by a configuration bit in the miscellaneous DCF client in UTest flash memory. The checker core shall always be configured to be enabled. If the LSM is disabled, the checker core and the RCCUs are disabled. The checker core will not work independently from the master core. No dynamic switching is possible between LSM on and LSM off, and a reset is required to reestablish LSM. Disabling of LSM triggers a fault indication to the FCCU.

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 MPC5777C.

The MPC5777C 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 MPC5777C, 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 MPC5777C Data Sheet in section "Absolute maximum ratings").

The external circuit will also monitor the ERROR*n* signals. Through a digital interface (for example, SPI), the MPC5777C repetitively triggers the watchdog of the external IC. If there is a recognized failure (for example, watchdog not being serviced, assertion of ERROR*n*), the reset output of the external IC will be asserted to reset the MPC5777C. 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).

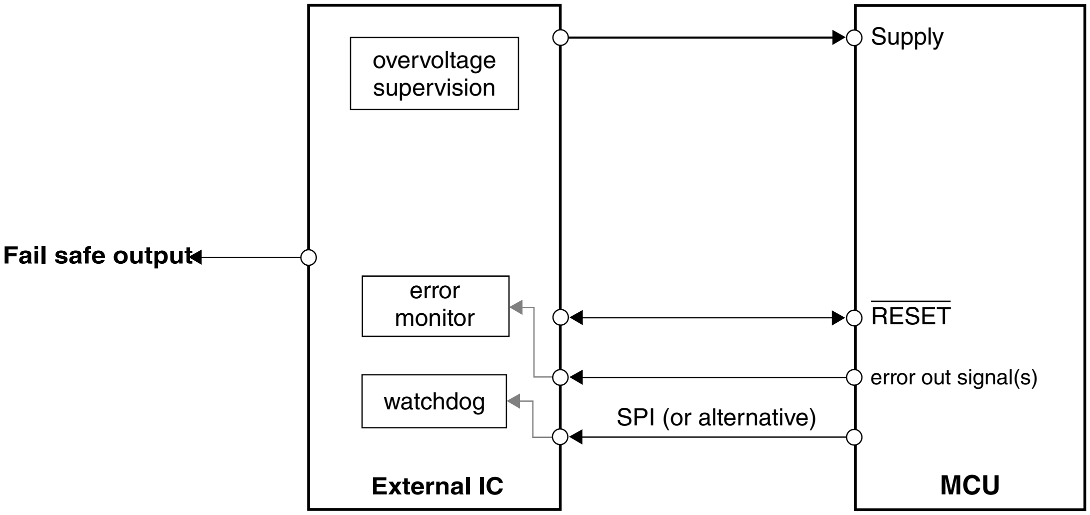


Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the MPC5777C for safety-related systems.

It should be noted that failure modes of external services are only partially considered in the FMEDA of the MPC5777C (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 MPC5777C is considered to be in a Safe state$_{\text{MCU}}$ (for example, unpowered and outputs tristated), the system containing the MPC5777C may not be compliant with the Safe state$_{\text{system}}$. A possible system level safety measure to achieve Safe state$_{\text{system}}$ may be to place pull-up or pull-down resistors on I/O when the high-impedance state is not considered safe.

Assumption: [SM_038] If a high-impedance state on an output pin is not safe, pull-up or pull-down resistors shall be added to safety-related outputs. The need for this will be application dependent for the unpowered or reset (tristated I/O) MPC5777C.[end]

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

4.1.1.2 External Watchdog (EXWD)

An external device, acting as an independent timeout functionality (for example, External Watchdog (EXWD)), should be used to cover Common Mode Failures (CMF) of the MPC5777C for safety-related systems.

The trigger may be a discrete signal(s) or message object(s). If within a defined timeout period the EXWD is not triggered, a failure will be considered to have occurred which would then switch the system to a Safe state$_{\text{system}}$ within the FTTI (for example, the EXWD disconnects the MPC5777C from the power supply, or communication messages are invalidated by disabling the physical layer driver).

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

The implementation of the communication between the MPC5777C and the EXWD can be chosen by the user as warranted by the application. Examples of different mechanisms that can be used to trigger the EXWD can include any of the following:

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

4.1.1.3 Power Supply Monitor (PSM)

Supply voltages outside of the specified operational ranges may cause permanent damage to the MPC5777C, even if it is held in reset.

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

The *MPC5777C Microcontroller Data Sheet* provides specific operating voltage ranges that must be maintained.

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

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

If the power supply is out of range, *MPC5777C* 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 *MPC5777C* even if kept in reset.

Implementation hint: An external and independent device may provide an over voltage monitor for the external MPC5777C supplies. If the supplied voltage supply is above the recommended operating voltage range of the MPC5777C, the MPC5777C should be maintained with no power. The external power supply monitor will switch the system to a Safe state$_{\text{system}}$ within the FTTI, and maintain it in Safe state$_{\text{system}}$ (for example, over-voltage protection with functional safety shut-off, or a switch-over to a second power supply unit).

If the MPC5777C 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 MPC5777C itself, but system level measures might be required to maintain the Safe state$_{\text{system}}$ in case an over-voltage situation may cause damage to the MPC5777C.

4.1.1.4 Error Out Monitor (ERRM)

If the MPC5777C signals an internal failure on its error out signals (ERROR0, and optionally ERROR1), the system may no longer rely on the integrity of the other MPC5777C outputs for safety functions. If an error is indicated, the system has to switch to, and remain in, Safe state$_{\text{system}}$ without relying on the MPC5777C. 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 ERROR*n* of the MCU and move the system to a Safe state$_{\text{system}}$ when an error is indicated. [end]

4.1.1.4.1 Both FCCU signals connected to separate device

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

This configuration does not require any dedicated software support.

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

NOTE

See “EOUT interface” section in the “Fault Collection and Control Unit (FCCU)” chapter of the *MPC5777C 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, ERROR0 (or ERROR1), is connected to a separate device.

If a fault occurs, the FCCU communicates the fault to the separate device through the ERROR0 (or ERROR1).

The functionality of ERROR0 (or ERROR1) can be checked in the following manner:

- ERROR0 (or ERROR1) read back internally.
- ERROR0 (or ERROR1) connected externally to a GPIO.
- ERROR0 (or ERROR1) uses time domain coding (for example, is active for a deterministic time interval).
- Test the ability of ERROR0 (or ERROR1) to disable system functionality (for example, measure voltage available at a motor if ERROR0 (or ERROR1) 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 ERROR*n* signal being used instead of using both ERROR*n* signals as in the previous section, is the lack of need for the separate device to compare the ERROR*n* signals.

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

Recommendation: If ERROR0, or ERROR1, is connected to a device not using time domain coding, verification is needed that the ERROR*n* signal(s) are operating correctly before execution of any safety function can start.

Rationale: To check the integrity of ERROR0, or ERROR1

To verify the functionality of a ERROR*n* signal, a fault may be injected into one of the ERROR*n* signals. The behavior of the signal can then be verified by the other ERROR*n* signal, or GPIO. Additionally, the fault output mode can be configured to one of the test modes to control one ERROR*n* as an output while the other ERROR*n* pin is an input or output. For example, TEST0 mode configures ERROR0 as an input and ERROR1 as an output. This test mode can be used to check the state of the ERROR0 input by reading FCCU_EINOUT[EIN0]. Likewise, the user can control the ERROR1 output by modifying FCCU_EINOUT[EOUT1].

Since the FCCU will be monitoring the system, it is sufficient to check ERROR0 (or ERROR1) within the L-FTTI (for example, at power-up) to help reduce the risk of latent faults. It is recommended that ERROR*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 ERROR0 (or ERROR1) 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 ERROR0 (or ERROR1) will result in the higher drive strength of ERROR*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 Pad Configuration Register (SIU_PCRn).

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 (ERROR0, or ERROR1), 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 MPC5777C 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 MPC5777C capable of performing input capture functionality.

The system must be switched to Safe $state_{system}$ if the MPC5777C 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 MPC5777C 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 MPC5777C 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 MPC5777C.

Out of a single battery supply with a wide voltage range (V_{SUP}, 3.5 V…28 V), the MC33907/08 generates 5 V (Vaux), 3.3V (Vcca), and 1.25V (Vcore) to supply the MPC5777C. 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 ERROR0 and ERROR1, using the bistable protocol. Via SPI, the MPC5777C 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 MPC5777C. 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 MPC5777C.

By a connection of the signal MUX_OUT to an ADC input of MPC5777C, 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 MPC5777C directly with a CAN or LIN bus.

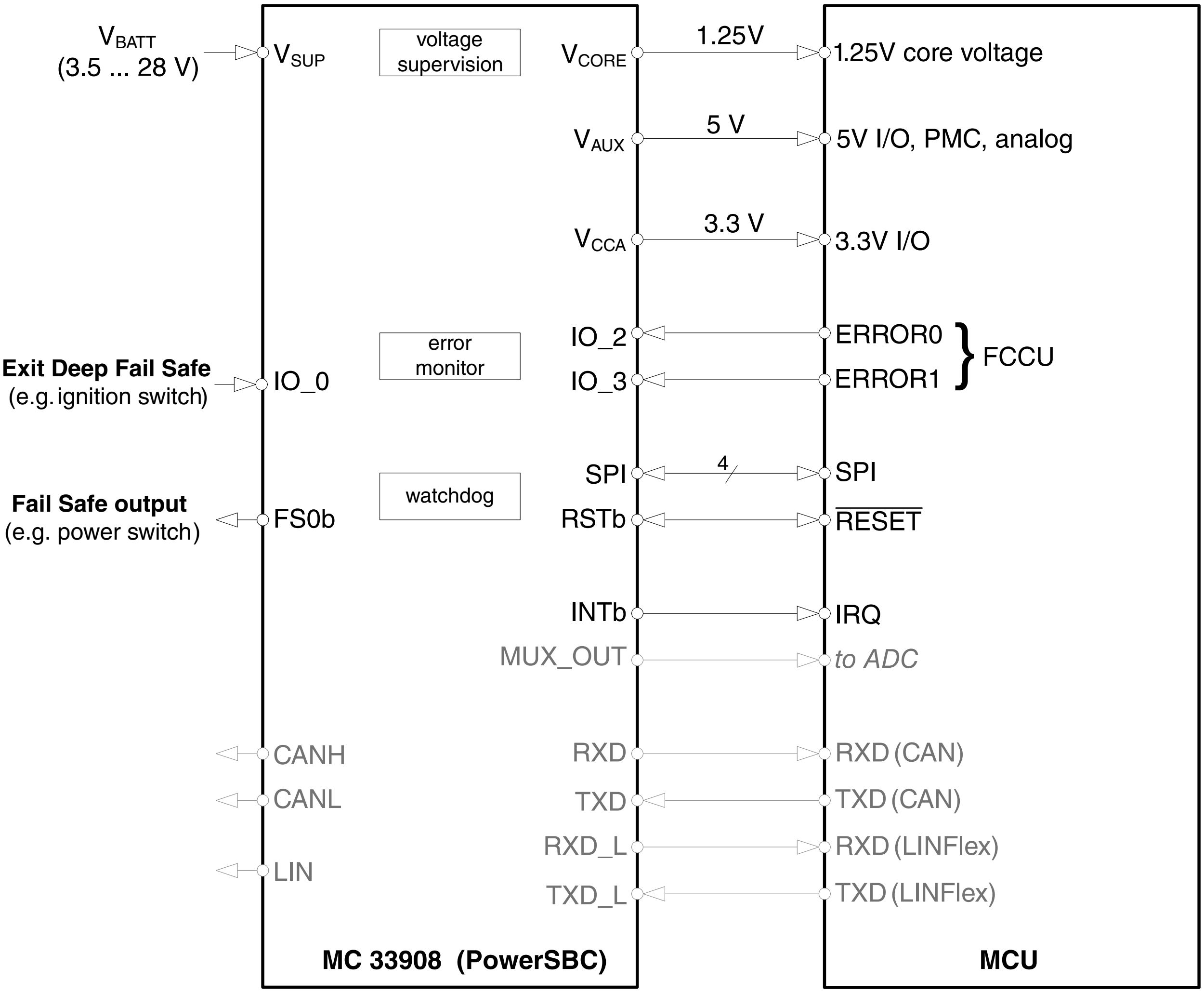


Figure 4-2. Functional safety application with PowerSBC

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, measures when using the individual components of the MPC5777C.

Given the application independent nature of the MPC5777C, no general safety function can be specified. To define a specific safety function, the MPC5777C would have to be integrated into a complete (application dependent) system. Nevertheless, it is possible to define abstract safety function elements and safety integrity functions:

- A safety function element is used to implement (or control) functional safety with available hardware.
- A safety integrity function (often called diagnostic measures) is to improve the probability of successful execution of functional safety.

Modules not explicitly covered by this document do not require safety-specific software measures. It is also possible to ignore the required measures for explicitly mentioned modules if equivalent measures to manage the same failures are alternatively included.

The modules that are replicated reach a very high diagnostic coverage (DC) without additional dedicated measures at application or system level.

5.1.1 Disabled modes of operation

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

5.1.1.1 Debug mode

The debugging facilities of the MPC5777C 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 MPC5777C may not enter Debug or HALT mode. The state of the JCOMP signal determines whether the system is being debugged or whether the system is operating in normal operating mode. When JCOMP is logic low, the JTAGC TAP controller is kept in reset for normal operating mode. When JCOMP is logic high, the JTAGC TAP controller is enabled to enter debug mode. During boot, measures shall be taken to ensure that JCOMP is not asserted by external sources, to avoid entering debug mode. The activation of debug mode, if JCOMP is low (for example, due to hardware failures), is supervised by the FCCU, and will signal a fault condition when debug mode is entered. If the FCCU recognizes erroneous activation of debug mode, JTAG signals will no longer recognize any input as being legal debug commands.

Assumption: [SM_147] Debugging shall be disabled in the field while the device is being used for safety-relevant functions. [end]

Assumption under certain conditions: [SM_148] In general for any module which can be frozen in debug mode and is 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, SDADC_CR[FRZ] controls operation of the SDADC. If the SDADC_CR[FRZ] = 0, the SDADC does not stop conversions in debug mode.

In debug mode, EQADC_MCR[DBG] controls operation of the EQADC. If the EQADC_MCR[DBG] = 0b00, the EQADC does not stop conversions in debug mode.

In debug mode, DECFILTER_MCR[FREN] controls operation of the DECFILTER module. If the DECFILTER_MCR[FREN] = 0, the DECFILTER module does not enter freeze mode when the device enters 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.

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

When EMIOS_MCR[FRZ] = 0, the eMIOS continues normal operation while the device is in debug mode. NOTE: There is a freeze mask bit within EMIOS_C register - EMIOS_C[FREN].

In debug mode, REACM2_MCR[FREN] controls operation of the Reaction Module. If the REAC2_MCR[FREN] = 0, the Reaction Module does not enter freeze mode when the device enters debug mode.

In debug mode, IGF_MCRn[FRZ] controls operation of the Input Glitch Filter Modules. If the IGF_MCRn[FRZ] = 0, the Input Glitch Filter does not enter freeze mode when the device enters debug mode.

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

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.

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.

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

In debug mode, PSI5_CHn_PCCR[DEBUG_EN] controls operation of the PSI5 channel. If the PSI5_CHn_PCCR[DEBUG_EN] = 0, the module will continue to operate normally in debug mode.

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

In debug mode, FCCU_CTRL[DEBUG] controls operation of the FCCU. If the FCCU_CTRL[DEBUG] = 0, the module continues to operate normally in debug mode.

5.1.1.2 Test mode

Several mechanisms of the MPC5777C can be circumvented during test mode which endangers the functional safety integrity.

Assumption: [SM_149] Test mode is used for comprehensive factory testing and is not valid for normal operation. Test mode shall be not be enabled in the field while the device is being used for safety-relevant applications. [end]

Implementation hint: The TEST pin is for test purposes only, and may be tied to GND during normal operating mode. From a system level point of view, measures may ensure that the TEST pin is not connected to V_{DD} during boot to avoid entering Test mode. The activation of Test mode by any source is supervised by the FCCU and may signal a fault condition when Test mode is entered.

5.2 MPC5777C modules

5.2.1 Mulitplexed serial communicaton protocol controllers

An appropriate safety software protocol should be utilized (for example, Fault-Tolerant Communication Layer, FTCOM) for any communication peripheral employed to meet high safety integrity level application requirements.

Recommendation: [SM_151] It is assumed that communication over the following interfaces is protected by a fault-tolerant communication protocol (implemented by the operating system or the application) in general in the OSI transport layer (Layer 4):

- FlexCAN
- MCAN
- Ethernet [end]

FlexCAN and MCAN do not have safety mechanisms other than what is included in their protocol specifications. It is assumed that communication over high-bandwidth interfaces is protected by a fault-tolerant communication protocol (implemented by the operating system or the application).

FlexCAN modules are redundantly available peripheral modules. FlexCAN_A and FlexCAN_B are connected to PBRIDGE_B, and FlexCAN_C and FlexCAN_D are connected to PBRIDGE_A.

5.2.2 Fault Collection and Control Unit (FCCU)

The FCCU uses a hardware safety channel 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 shall 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 redundant hardware checker results (for example, the RCCU, see Redundancy Control Checking Unit (RCCU))
- Collection of error information from modules whose behavior is essential to the functional safety function
- Configurable and graded fault control:
  - Internal reactions
    - No fault indication (safe fault)
    - Maskable and non-maskable interrupts
    - Resets
  - External reaction (external failure reporting using ERROR*n*)

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

The "FCCU fault inputs" table shown in section "Fault Collection and Control Unit (FCCU)" of the *MPC5777C Reference Manual* lists sources of critical faults that are signaled to the FCCU and the type of reset asserted.

The FCCU has two external signals, ERROR0 and ERROR1, through which critical failures are reported. When the device is in reset or unpowered, these outputs are tristated.

ERROR*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 in a Safe state$_{system}$ condition within the FTTI (for example, the separate device disconnects the MPC5777C or an actuator from the power supply).

5.2.2.1 Initial checks and configurations

Other than the possible initial configuration, no intervention from the MPC5777C is necessary for fault collection and reaction.

Assumption: [SM_153] Before starting safety-relevant operations, software shall ensure that the fault reaction to each safety-relevant fault is configured. [end]

Rationale: To maintain the device in the Safe state$_{\text{system}}$ in case of failure.

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

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

Assumption: [SM_166] If the MPC5777C signals an internal failure via its error out signals (ERROR*n*), the system can no longer safely use the MPC5777C safety function outputs. If an error is indicated, the system has to be able to remain in Safe state$_{\text{system}}$ without any additional action from the MPC5777C. Depending on its functionality, the system might disable or reset the MPC5777C as a reaction to the indicated error. [end]

5.2.2.2 Runtime checks

Assumption under certain conditions: [SM_155] If the MPC5777C is continuously switching between different Safe state$_{\text{MCU}}$, without a device shutdown, system level measures shall be implemented to ensure that the system meets the Safe state$_{\text{system}}$ criteria. [end]

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

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

The application software shall store previous FCCU error indications. If several consecutive resets are caused by the same FCCU error, the application software should signal a failure.

Assumption: [SM_248] Before resetting the reset counters, the application software shall ensure that it can detect longer reset cycles caused by faults in normal operation. [end]

5.2.3 Self Test Control Unit (STCU2)

The STCU2 executes BISTs (LBIST, MBIST) and reacts to detected faults by signaling faults to the FCCU (see "Self-Test Control Unit (STCU2)" in the *MPC5777C Reference Manual* for details).

5.2.3.1 Initial checks and configurations

Application software is not required to configure the STCU2. It is assumed that LBISTs and MBISTs are executed once per T_{trip}.

Recommendation: [SM_162] When Built-In Self-Test (for example, LBIST, MBIST) circuits of the MPC5777C are used as functional safety integrity measures (for example, to detect random faults, latent faults, and single-point faults) in a functional safety system, functional safety integrity measures on a system level may 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: System (application) level software shall check the STCU2 to ensure integrity.

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

This software confirmation prevents a fault within the STCU2 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 FCCU. So, reading STCU2_LBS, STCU2_LBE, STCU2_MBSL, STCU2_MBSM, STCU2_MBSH, STCU2_MBEL, STCU2_MBEM, STCU2_MBEH and STCU2_ERR registers increases the STCU2 fault coverage.

Implementation hint: The STCU2 may 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 (STCU)" chapter of the *MPC5777C Reference Manual*).

5.2.4 Temperature Sensors (TSENS)

The MPC5777C has two temperature sensors that are read from the EQADC_A module (ADC0 and ADC1 converters on EQADC_A channels 128 and 129). 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 application software level.

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

- 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 threshold detection feature is enabled, the temperature sensor monitors the internal junction temperature of the chip at two different spatial locations and asserts a signal if any of the specified temperature thresholds are crossed (as shown in section "Temperature threshold detection (digital output generation)" of the *MPC5777C Reference Manual*).

Note

The threshold temperatures may be adjusted by configuring the "Temperature detector configuration register (PMC_CTL_TD)" located in the PMC module.

The temperature sensors monitor the substrate temperature to detect overtemperature conditions before they cause CMFs (for example, faults due to overtemperature which causes identical erroneous results from both cores). The maximum operating junction temperature is specified in the *MPC5777C Data Sheet*. The sensor output is forwarded to the appropriate EQADC channels for measurement conversion.

5.2.4.1 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_164] 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 (see the "FCCU fault inputs" table shown in section "Fault Collection and Control Unit (FCCU)" of the *MPC5777C Reference Manual*). [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 the internal temperature sensors.

Assumption under certain condition: [SM_166] If latent fault diagnostic coverage of temperature sensing is safety relevant for an application, during power up the two temperature sensors are to be read by software. The software shall verify that the conversion values are similar, as this is a means of assessing that the sensors are working properly. [end]

5.2.5 Software Watchdog Timer (SWT)

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 shall 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 the "FCCU fault inputs" table shown in section "Fault Collection and Control Unit (FCCU)" of the *MPC5777C 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_102] 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 register, the SWT control register flags SWT_CR[WEN] and SWT_CR[HLK] need to be asserted.

Note

> The timeout register (SWT_TO) shall contain a 32-bit value that represents a timeout less than the FTTI/PST.

In general, it is expected that application software uses the SWT to detect lost clocks or clocks that are too slow. 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 MPC5777C provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudo-random key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

Assumption: [SM_169] 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:

- The first one is based on the fix service sequence represented by a write of SWT_KEY1 value followed by a write of SWT_KEY2 value to the SWT service register. Writing the service sequence re-loads the internal down counter with the time-out 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 $SK_{(n+1)} = (17 \times (SK_n + 3)) \bmod 2^{16}$.

5.2.5.1 Run-time checks

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

5.2.6 Redundancy Control Checking Unit (RCCU)

The task of the Redundancy Control Checking Unit (RCCU) unit is to perform a cycle-by-cycle comparison of the outputs between core 1 and the checker core. The error information is forwarded to the FCCU. The RCCUs are automatically enabled when the MPC5777C Lockstep feature is enabled.

NOTE

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

5.2.6.1 Initial checks and configurations

The use of the RCCU is indispensable, and is automatically managed by the MPC5777C. The RCCU cannot be disabled by application software. Consequently, the respective FCCU input should not be disabled.

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]

5.2.7 Cyclic Redundancy Checker Unit

The Cyclic Redundancy Checker Unit (CRC) offloads the CPU in computing a CRC checksum. The CRC has the capability to process two interleaved CRC calculations. The CRC module may be used to detect erroneous corruption of data during transmission or storage. The CRC takes as its input a data stream of any length and calculates a 32-bit output value (signature). There are three sets of CRC registers to allow concurrent CRC computations in the MPC5777C.

The contents of the configuration registers of the functional safety related modules shall be checked within the FTTI.

5.2.7.1 Runtime checks

Parts of the MPC5777C configuration registers do not provide the functional safety integrity IEC 61508 series and ISO 26262 require for high functional safety integrity targets. This relates to systematic faults (for example, application software incorrectly overwriting registers), as well as random hardware faults (bit flipping in registers).

Assumption: [SM_170] The CRC calculation shall be executed at least once per FTTI to verify the content of the safety-relevant configuration registers. [end]

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

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

Assumption:[SM_171] Safety software running on the safety core must check correct initialization of the MPC5777C before activating the safety-relevant functionality.[end]

Note

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

Implementation hint: To verify the content of the MPC5777C configuration registers of the modules involved with the safety function, the CRC module may be used to calculate a signature of the content of the registers and compare this signature with a value calculated during development.

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

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

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

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

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

5.2.7.1.1 Implementation details

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

Note

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

5.2.7.1.1.1 *<module>*_SWTEST_REGCRC

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

- EMIOS0_SWTEST_REGCRC

  The eMIOS_0 configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- EMIOS1_SWTEST_REGCRC

The eMIOS_1 configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- SIU_SWTEST_REGCRC

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

- ETPUA_SWTEST_REGCRC

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

- ETPUB_SWTEST_REGCRC

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

- ETPUC_SWTEST_REGCRC

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

- EQADC_A_SWTEST_REGCRC

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

- EQADC_B_SWTEST_REGCRC

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

- SDADC0_SWTEST_REGCRC

  The SDADC0 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- SDADC1_SWTEST_REGCRC

  The SDADC1 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- SDADC2_SWTEST_REGCRC

  The SDADC2 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- SDADC3_SWTEST_REGCRC

The SDADC3 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

5.2.8 Internal RC oscillator (IRCOSC)

The Internal RC oscillator (IRCOSC) has a nominal frequency of 16 MHz, but the frequency accuracy over the full voltage and temperature range has to be taken into account. Functional safety-related modules which use the clock generated by the IRCOSC are: FCCU, CMU, and SWT. In the rare case of an IRCOSC clock failure, these modules will stop functioning.

5.2.8.1 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_173] 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 CMU_CSR[SFM] = 0 the frequency measurement has completed (see "Frequency meter" section in the "Clock Monitor Unit (CMU)" chapter of the *MPC5777C 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.

5.2.8.2 Runtime checks

The frequency meter 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_174] 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]

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$_{\text{system}}$ 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.

5.2.9 External Oscillator (XOSC)

FlexCAN, MCAN, and PIT each feature modes in which they are directly clocked from the XOSC. For safety relevant applications, these clocking modes should not be used.

5.2.9.1 Initial checks and configurations

Assumption: [SM_175] The application software shall not utilize, for safety-relevant applications, FlexCan, MCAN, and PIT modules in modes in which the modules are clocked directly from the XOSC. [end]

5.2.9.2 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 to the operational state.[end]

[1] Nominal frequency of the IRCOSC is 16 MHz, but the post trim accuracy over voltage and temperature shall be taken into account.

5.2.10 Dual PLL Digital Interface (PLLDIG)

The MPC5777C 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 FCCU. 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 MCU modules. The auxiliary clock from the non-FMPLL is distributed to those peripherals that require precise timing.

Implementation hint: PLLDIG_PLL0SR[LOLF] and PLLDIG_PLL1SR[LOLF] indicate 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.

5.2.10.1 Initial checks and configurations

After system reset, the PLLs are deactivated. The MPC5777C initially uses the internal RC oscillator clock (IRCOSC) as clock source (see the "Clocking" chapter in the *MPC5777C Reference Manual* and Internal RC oscillator (IRCOSC) for details on IRCOSC configuration).

Assumption: [SM_178] 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 MPC5777C. [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 externally provided clock reference.

Assumption under certain conditions: [SM_179] When clock glitches endanger the system level functional safety integrity measure or functional safety-relevant modules, or both, they 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 from the external crystal and its hardware connection to the MPC5777C.

Implementation hint: This requirement is fulfilled by appropriately programming the System Integration Unit (SIU).

During/after initialization but before executing any safety function, application software has to check that the MPC5777C uses the FMPLL clock as "system clock".

Implementation hint: Application software can check the current system clock by checking the SIU_SYSDIV[SYSCLKSEL] bit field. SIU_SYSDIV[SYSCLKSEL] = 2 indicates that the FMPLL (PLL1) clock is being used as the system clock.

5.2.11 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 MPC5777C is still running on the IRCOSC.

Clocks are supervised by Clock Monitoring Units (CMUs). The CMUs are driven by the 16MHz internal reference clock oscillator (IRCOSC) to ensure independence from the monitored clocks. CMUs detect errors associated with conditions due to clock out of programmable bounds or loss of clock. If a supervised clock leaves the specified range for the device, an error signal is sent to the FCCU. MPC5777C includes the CMUs shown in Table 5-1.

Table 5-1. Clock Monitoring Units

CMU	Monitored Clock
CMU_0	PLL0,XOSC,IRCOSC
CMU_1	Core 0, Core 1, Checker Core, RCCU
CMU_2	Platform modules: Crossbar, Peripheral Bridges, Memories, DMA, Flash Memory Controller, Debug
CMU_3	Memory-mapped registers of peripherals
CMU_5	Protocol and communication engines of peripherals
CMU_6	SDADC
CMU_7	PSI5 Rx clock input
CMU_8	PSI5 1 MHz clock input

The CMUs are programmable to allow them to:

- detect clock out of a programmable frequency range (frequency too high or too low)
- adjust the time over which the supervised clock is monitored for a frequency violation

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 fault inputs" table shown in section "Fault Collection and Control Unit (FCCU)" of the *MPC5777C Reference Manual*).

Assumption: [SM_180] For safety-relevant applications, CMU use is mandatory. If the modules monitored by the CMU 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 CMU faults, so it shall be configured accordingly. [end]

5.2.11.1 Initial checks and configurations

Assumption: [SM_181] The following supervisor functions are required:

- Loss of external clock
- FMPLL frequency higher than the (programmable) upper frequency reference
- 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 reference (CMU_*n*_HFREFR_A[HFREF_A]) and low frequency reference (CMU_*n*_LFREFR_A[LFREF_A]) is configured depending on CMU_0.

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

5.2.12 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. The module has low voltage detectors (LVD) and high voltage detectors (HVD). If one of the monitored voltages goes below (LVD) or above (HVD) a given threshold, a reset is initiated to control erroneous voltages before these cause a CMF (see the *MPC5777C Data Sheet* for correct operating voltage ranges).

To ensure functional safety, the Power Management Controller (PMC) monitors various supply voltages of the MPC5777C device (as seen in Table 5-2):

- The low and high voltage detectors (LVD/HVD) supervise the 1.25 V core supply (VDD_LV) voltage to verify that it maintains a level between the lower and upper limits.
  - VDD LVD
  - VDD HVD

Assumption: [SM_244] The application software shall initiate the LVD/HVD self-test mechanism to detect LVD/HVD failures after startup.[end]

Assumption: [SM_184] The application software shall check the status registers of the FCCU 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 because the operation of the PMC is automatic.

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

Undervoltage and overvoltage conditions can directly cause a transition into a safe state via a reset. This solution was chosen because safety-relevant voltages have the potential to disable the failure indication mechanisms of the MPC5777C (the FCCU). The LVDs and HVDs also report errors to the FCCU, but since the LVD and HVD errors can result in reset, the FCCU error reporting is not utilized.

Note

> For development purposes only, different fault reactions can be programmed in the PMC for LVD and HVD error reporting to the FCCU and reset/interrupt generation can be disabled.

Assumption: [SM_185] Software shall not disable the direct transition by reset into a safe state due to an overvoltage or undervoltage indication.[end]

Table 5-2. PMC monitored supplies

Detector Type	Detector Name	Voltage Monitored
LVD	PMC voltage supply	3.3 - 5.0 V PMC supply
POR/LVD/HVD	VDD core voltage supply	1.25 V core supply
LVD/HVD	Flash memory voltage supply	3.3 V flash supply
LVD	VDDIO I/O voltage supply	5 V I/O supply

NOTE

See the *MPC5777C Data Sheet* for voltage limits of these supplies

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

5.2.12.1 1.25 V supply supervision

Voltage detectors LVD_core and HVD_core monitor the digital (1.25 V) core supply voltage for over and under voltage in relation to a reference voltage. In case the core main voltage detector detects over or under voltage during normal operation of the MPC5777C, a reset is triggered.

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

Assumption under certain conditions:[SM_189] 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, overvoltage can not be shut off redundantly. [end]

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

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

5.2.12.2 3.3 V supply supervision

Voltage detectors LVD_FLASH and HVD_FLASH monitor the 3.3 V flash supply for under voltage and over voltage in relation to a reference voltage. In case a single LVD/HVD detects under/over voltage during normal operation of the MPC5777C, a reset is triggered. Note that if the internal flash regulator is used, it guarantees the correct operating range for the flash. LVD/HVD may be triggered if external 3.3V is supplied to both VDDPMC and VDDFLA, bypassing the internal flash regulator and if that external supply does not maintain the correct operating range for the flash at all times.

5.2.13 Memory Protection Units (MPU)

As a multimaster, concurrent bus system, the MPC5777C provides safety mechanisms to prevent non-safety masters from interfering with the operation of the safety core. The MPC5777C also contains mechanisms to handle the concurrent operation of software tasks with different or lower ASIL classifications.

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

Assumption: [SM_192] The MPU shall only be programmed by the safety core. This software shall prevent write accesses to the MPU's registers from all other masters. The MPU programming model shall only be accessible to the safety core. [end]

5.2.13.1 Initial checks and configurations

Assumption under certain conditions:[SM_195] If nonreplicated bus masters (for example, CSE, SIPI, and FEC) are used, system level functional safety integrity measures shall cover bus operations to reduce the likelihood of replicated resources being erroneously modified. [end]

Rationale: Access restriction is protection against unwanted read/write accesses to some predefined memory mapped address locations.

Implementation hint: The MPU shall be used to ensure that only authorized software routines can configure modules and all other bus masters (CSE, SIPI, FEC, eDMA) can access only their allocated resources according to their access rights.

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

Recommended: The MPU may be used to ensure that only authorized software routines (processes) can configure modules and access private resources. All other software routines can access only their allocated resources according to their access rights.

5.2.14 PBRIDGE protection

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

- Master privilege level – The access privilege level associated with each master is configurable. Each master can be configured to be trusted for read and write accesses.
- Peripheral access level – The access level of each peripheral is configurable. The peripheral can be configured to require the master accessing the peripheral to have supervisor access attribute. Furthermore, if 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.

5.2.14.1 Initial checks and configurations

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

5.2.15 Built-In Hardware Self-Tests (BIST)

Built-in hardware self-tests (BISTs) or built-in tests (BITs) are mechanisms that permit circuitry to test itself. Hardware supported BIST is used to speed up self-tests and reduce CPU load. As hardware assisted BIST is often destructive, it shall be executed ahead or after a reset.

Absence of latent faults shall be checked at startup or during shutdown by MBIST or LBIST. The boot time BIST includes the scan-based LBIST to test the digital logic and the MBIST to test all RAMs and ROMs.[2]

[2] This does not include flash memory.

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, or destructive reset, and it will also execute when initiated by software (online).

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 shall decide whether to continue or halt execution. This is true even if several of the correctable failures combine 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_197] 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]

Note

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

- STCU2_LBS to determine which offline LBISTs failed
- STCU2_LBE to determine which offline LBISTs did not finish
- STCU2_MBSL, STCU2_MBSM and STCU2_MBSH to determine which offline MBISTs failed
- STCU2_MBEL, STCU2_MBEM and STCU2_MBEH to determine which offline MBISTs did not finish
- STCU2_LBSSW to determine which online LBISTs failed
- STCU2_LBESW to determine which online LBISTs did not finish
- STCU2_MBSLSW, STCU2_MBSMSW and STCU2_MBSHSW to determine which online MBISTs failed
- STCU2_MBELSW, STCU2_MBEMSW and STCU2_MBEHSW – To determine which online MBISTs did not finish
- STCU2_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 period of time (latent faults), they can accumulate once they propagate. ISO 26262 requires 90% latent-fault metric for ASIL D, 80% for ASIL C, and 60% for ASIL B. Typically, hardware assisted BIST is therefore used as a safety integrity measure to detect latent faults.

The MPC5777C is equipped with a Built-in hardware self-test:

- System SRAM (MBIST, executed at boot-time, latent failure measure)
- Logic (LBIST, executed at boot-time, latent failure measure)
- Flash memory integrity self check (executed at least once per FTTI, single-point failure measure)
- Flash memory margin read (executed after every programming operation or executed at least once per FTTI, latent failure measure and single-point failure measure)
- PMC (self-test of LVD/HVD)

Boot-time tests (MBIST, LBIST) are performed after the occurrence of a power-on, FOSU, or external reset, unless they are disabled. All boot-time tests are executed before application software starts executing. If a boot-time test fails, the MPC5777C will remain in Safe state$_{\text{MCU}}$.

All tests may be performed without dedicated external test hardware.

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

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

5.2.15.1 Memory Built-In Self-Test (MBIST)

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

NOTE

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

5.2.15.2 Logic Built-In Self-Test (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.

5.2.15.3 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 Flash memory).

5.2.15.4 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.

5.2.15.5 Flash memory ECC logic check

The flash memory ECC logic check runs in flash memory user test mode. It is executed in software and supported by hardware.

5.2.15.6 Flash memory ECC fault report check

The flash memory ECC fault report check is executed in software (refer to Flash memory).

5.2.16 End-to-end ECC (e2eECC)

The MPC5777C 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 MPC5777C e2eECC protected memory, the bus master initiates the data write and generates ECC checkbits based on address and data. 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 the 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
- Every memory with ECC
    - ECC bits are stored alongside data in Flash memory and RAM. This includes flash memory array, RAM array, CAN RAM, DMA RAM, and eTPU RAM.
- ECC on address and data

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.

The ECC for RAM, without inclusion of address, marks 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.

5.2.17 Interrupt Controller (INTC)

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

Assumption: [SM_198] Application software will detect the critical failure modes of the INTC for all interrupts. [end]

Note

> **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.

5.2.17.1 Periodic low latency IRQs

The SWT 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_199] Periodic low latency IRQs will use a running timer/counter to ensure their call period is expected.[end]

5.2.17.2 Non-Periodic low latency IRQs

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

Recommendation: 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.

5.2.17.3 Runtime checks

Assumption under certain conditions: [SM_200] 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.

5.2.18 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.

The eDMA is not replicated, therefore it is assumed that it will not be used for transferring safety-critical data. Failures outside the eDMA can lead to the eDMA behaving incorrectly. Such failures must be detected by software.

5.2.18.1 Runtime checks

Assumption: [SM_201] The eDMA will be supervised by software which detects spurious, too frequent, or constant activation.[end]

Rationale: To prevent the eDMA from stealing transfer bandwidth on the XBAR, as well as preventing it from copying data at the wrong time.

Implementation hint: Possible software implementations to protect against spurious or missing interrupts are as follows:

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

Assumption under certain conditions:[SM_202] Applications that are not resilient to spurious, or missing functional safety-relevant, eDMA requests can not use the PIT module to trigger functional safety-relevant eDMA transfer requests. [end]

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

5.2.19 System timer module (STM)

5.2.19.1 Runtime checks

In case a failure in the System Timer Module (STM) causes a violation of the safety goal, then application software measures shall be employed to detect a stopped STM or one running with the wrong frequency.

Implementation hint: In the first option, the SWT can be configured to measure the time between STM interrupts and compare with the STM measured time. In the second option, application software inserts control-flow checkpoints in the STM IRQ handler and writes two pseudorandom keys to service the watchdog.

Assumption: [SM_205] At every STM interrupt, the IRQ handler shall compare the elapsed time since the previous interrupt to a free running counter to check whether the interrupt time is consistent with the STM setting. [end]

Assumption: [SM_206] The STM IRQ handler shall be under SWT protection.[end]

5.2.20 Periodic Interrupt Timer (PIT)

5.2.20.1 Runtime checks

Assumption: [SM_107]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

Recommendation under certain conditions: [SM_208] If the PIT is used in a safety-relevant application, a checksum of its configuration registers using the CRC shall be calculated and compared with the expected PIT configuration to verify correct settings. [end] The application software shall invoke this test once per FTTI/PST.

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

5.2.21 Flash memory

The MPC5777C provides programmable non-volatile (NVM) flash memory with ECC, which can be used for instruction and/or data storage.

The flash memory array integrity self-check detects possible latent faults affecting the flash memory array, including potential data retention issues or the logic involved in read operations. The array integrity self-check calculates a MISR signature over the array content to validate the content of the array as well as the decoder logic. The calculated MISR value depends on the array content and shall be validated by application software.

Implementation hint: The array integrity self check and the ECC logic check may be executed on each program flash memory block used.

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

5.2.21.1 EEPROM

MPC5777C provides blocks of the flash memory for EEPROM emulation. ECC events detected on accesses to the EEPROM flash memory blocks are not reported to the ERM. Single bit errors are corrected but not reported. Multi-bit errors are replaced by a fixed word (representing an illegal instruction) and are also not reported to the ERM.

Assumption:[SM_114] The software using the EEPROM emulation 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.

5.2.21.2 Initial checks and configurations

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: the test result may be validated by reading the corresponding registers in the flash memory controller after the test is complete (see the "Array integrity self check" section in the "Flash memory" chapter of the *MPC5777C Reference Manual*).

5.2.21.3 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_216] 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_217] 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 to confirm that the programmed data matches the data written.. 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_219] 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: It consists of reading a set of data words from flash memory having erroneous ECC bits programmed. Respective ERM register content may be validated by software.

5.2.22 Error reporting path tests

It is possible to check the correct operation of several reporting paths from supervisors to the ERM. The FCCU input table specifically lists these in the "FCCU Non-Critical Faults Mapping" table shown in the "Chip-specific FCCU Information" section of the *MPC5777C Reference Manual*.

Other measures in that column (except LBIST) can also be used for a full error reporting path check. 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 path checks can also be used during development to test whether software programmed to handled such faults works correctly.

Additionally, ECC errors can be injected into FlexCAN SRAM and System SRAM to check the reporting of such errors through the ERM to the FCCU.

A multiple cell failure caused, for example, by a neutron or alpha particle or a short circuit between cells may cause three or more bits to be corrupted in an ECC protected word. As a result, either the availability 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 the MPC5777C by the scrambling (column multiplexing), which means 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. The MPC5777C has a multiplexor factor of eight for its system RAM multiplexing adjacent analog bit lines to an analog sense amplifier. It is always enabled and needs no configuration.

5.2.23 Glitch filter

A glitch detector is implemented on the reset signal of the MPC5777C. A selectable (SIU_IDFR[DFL]) glitch filter is implemented on the IRQ-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.

5.2.24 Crossbar Switch (XBAR)

The multi-port XBAR switch allows for concurrent transactions from any master to any slave. 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 affect software execution times, so software countermeasures shall detect these faults.

Assumption: [SM_227] Masters of the XBar which are not safety-related modules shall have a lower arbitration priority on the XBar than safety-relevant masters. [end]

5.2.24.1 Runtime checks

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

The detection of failures of the XBAR configuration can be achieved by 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 stringently the control flow monitoring is implemented.

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

Assumption: [SM_228] Within the FTTI, application software shall detect failures of the XBAR configuration affecting system performance.[end]

5.2.25 Sigma-Delta Analog to Digital Converter (SD-ADC)

Portions of the Sigma-Delta Analog-to-Digital Converter (SD-ADC) of the MPC5777C do not provide the functional safety integrity that IEC 61508 series and ISO 26262 require for high functional safety integrity targets. Therefore, system level measures are required.

5.2.25.1 Initial checks and configurations

Assumption under certain conditions: [SM_249] When the SD-ADC of the MPC5777C is used in a safety function, suitable system level functional safety integrity measures shall be implemented after reset (external reset or power-on reset) before starting the respective safety function to ensure SD-ADC integrity. [end]

Rationale: To check the integrity of the SD-ADC function against latent failures

Implementation hint: After reset (external reset or power-on reset), but before executing any safety function, perform the gain and offset calibration of the SD-ADC using application software to detect latent faults.

5.2.26 Enhanced Queued Analog to Digital Converter (eQADC)

Portions of the Enhanced Queued Analog-to-Digital Converter (eQADC) of the MPC5777C do not provide the functional safety integrity that IEC 61508 series and ISO 26262 require for high functional safety integrity targets. Therefore, system level measures are required.

5.2.26.1 Initial checks and configurations

Assumption under certain conditions: When the eQADC of the MPC5777C is used in a safety function, suitable system level functional safety integrity measures shall be implemented after reset (external reset or power-on reset) before starting the respective safety function to ensure eQADC integrity.

Rationale: To check the integrity of the eQADC function against latent failures.

Implementation hint: After reset (external reset or power-on reset), but before executing any safety function, perform the gain and offset calibration of the eQADC using application software to detect latent faults.

5.3 I/O functions

The integrity of the peripheral subsystem will be mainly 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). System integrators may choose the approach that best fits their needs.

Assumption: [SM_233] Comparison of redundant operation of I/O modules is the responsibility of the application software. [end]

Assumption under certain conditions: [SM_234] No specific hardware measures have been implemented to specifically reduce CMFs with respect to replicated I/O peripherals. If the system level requires specific robustness regarding common mode faults within the I/O peripheral system, respective measures are required at the system level.[end]

Rationale: To improve the common mode fault robustness of the I/O modules.

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 use different replicated peripherals (for example, eMIOS_0 and eMIOS_1) to implement multiple independent and different channels.

5.3.1 Digital inputs

Assumption under certain conditions: [SM_237] When safety functions use digital inputs, system level functional safety mechanisms have to be implemented to achieve the required functional safety integrity level. [end]

5.3.1.1 Hardware

Implementation hint: Functional safety digital inputs may 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).

- The double read operation of a digital input is implemented by two general purpose inputs (GPI) of the SIU unit. A comparison (by software) between the double reads (for example, reads from both GPIOs) detects an error (please refer to Figure 5-1).
- 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 eMIOS_1 and the other by eMIOS_0. 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 5-1).

For each signal of a double read, the SIU can provide additional channels to support interrupt-based reading for each signal (see Figure 5-2).

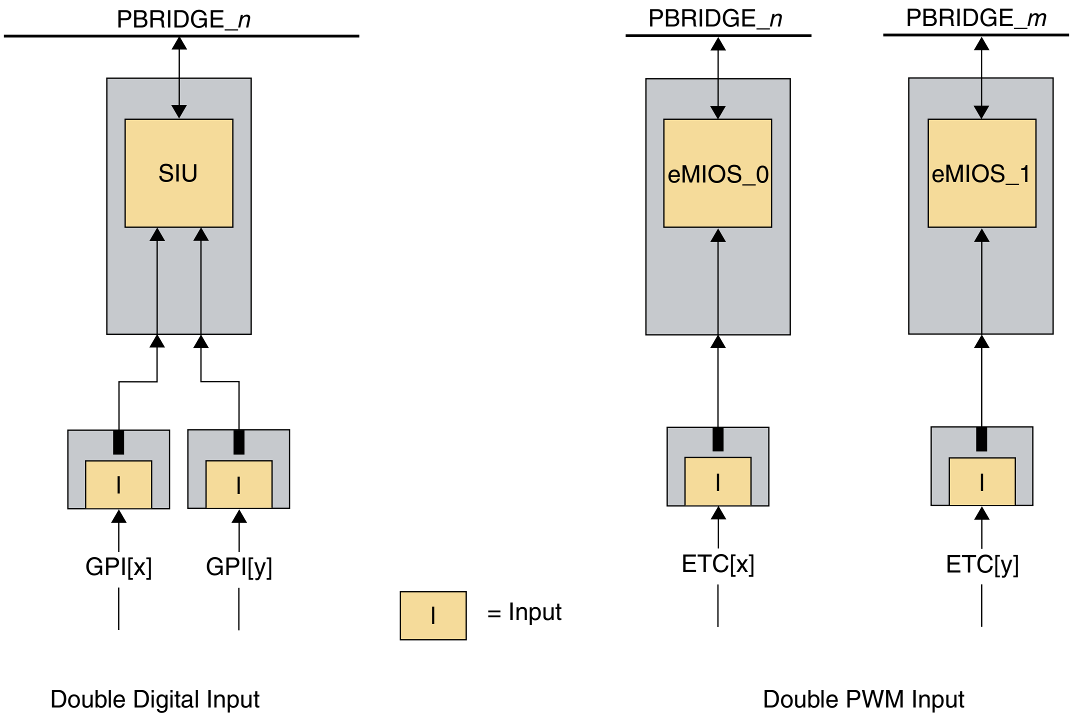


Figure 5-1. Double Digital input and Double PWM input

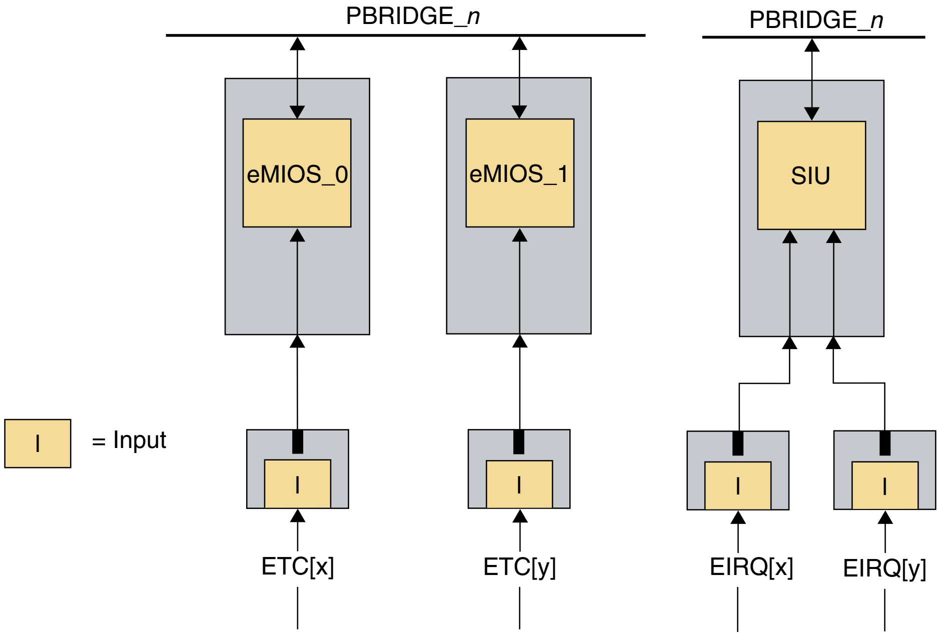


Figure 5-2. Double Read Encoder Input (IRQ triggered)

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

5.3.1.2 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, and 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	SIU_SWTEST_REGCRC	Once after programming
	GPI_SWTEST_CMP	Once for every acquisition
Double Read PWM Inputs	EMIOS0_SWTEST_REGCRC	Once after programming
	EMIOS1_SWTEST_REGCRC	Once after programming
	SIU_SWTEST_REGCRC	Once after programming
	EMIOS_SWTEST_CMP	Once for every acquisition

5.3.1.2.1 Double read digital inputs

Rationale: To check that the configuration of the two I/Os used corresponds 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 SIU_SWTEST_REGCRC and GPI_SWTEST_CMP.

5.3.1.2.1.1 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 the 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.

5.3.1.2.1.2 SIU_SWTEST_REGCRC

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

5.3.1.2.1.3 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.

5.3.1.2.2 Double read PWM inputs

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

Rationale: To check that the configuration of the modules used by this safety function match 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:

- EMIOS0_SWTEST_REGCRC
- EMIOS1_SWTEST_REGCRC
- SIU_SWTEST_REGCRC

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

- EMIOSI_SWTEST_CMP.

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

5.3.1.2.2.1 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:

- eMIOS_0 channels
- eMIOS_1 channels

The system integrator may select one channel from eMIOS_0 and another from eMIOS_1.

5.3.1.2.2.2 EMIOSx_SWTEST_REGCRC and SIU_SWTEST_REGCRC

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

5.3.1.2.2.3 EMIOSI_SWTEST_CMP

This test is used to execute the comparison between the double reads of PWM inputs performed by two channels of different eMIOS (for example, eMIOS_0 and eMIOS_1). The comparison may take into account possible approximation because of different capturing of the asynchronous input signals.

5.3.2 Digital outputs

Functional safety digital outputs are always assumed to be written either redundantly or with read back. In the case of a single output with read back, the feedback loop should be as large as possible to cover faults on the system level. The figure below depicts the connection of two (functional safety critical) actuators connected to the MPC5777C. 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 eMIOS. This measure is to confirm that the generated output signal is correct. This read back may be internally of the MPC5777C (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 desired output connection is part of the I/O functional safety concept on system level.

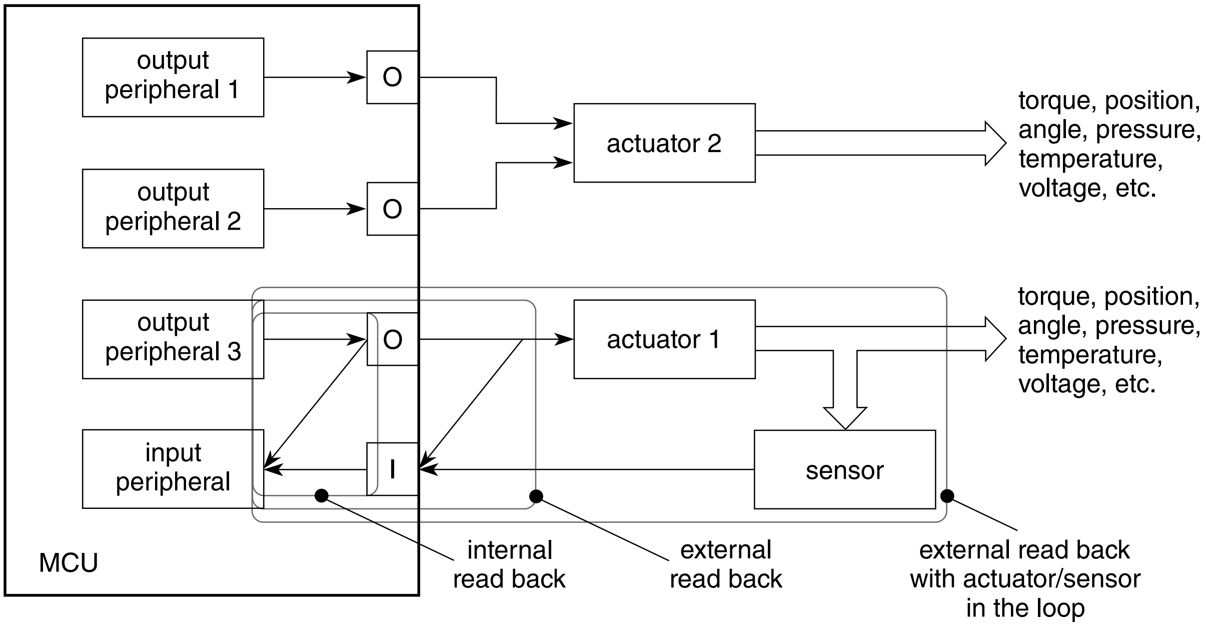

output peripheral 1
output peripheral 2
output peripheral 3
input peripheral
O
O
O
I
MCU
actuator 2
actuator 1
sensor
torque, position, angle, pressure, temperature, voltage, etc.
torque, position, angle, pressure, temperature, voltage, etc.
internal read back
external read back
external read back with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

Figure 5-3. 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 **Assumption** deviation as described in Safety manual assumptions.

5.3.2.1 Hardware

5.3.2.1.1 Single Write Digital Output

- Single Write Digital Output with external read-back (Figure 5-4, left):

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

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

  A comparison between the desired output values and the value read back is executed 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 5-5, left):

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

This procedure compares the output of the PWM read-back provided by a single channel of the eMIOS with the expected values that have been written to the external pad of the eTPU output channel.

- Single Write PWM Output with internal read-back[3] (Figure 5-5, right):

  This procedure output compares the PWM read-back by a single channel of the eMIOS with the expected values that have been written to the eTPU output channel.

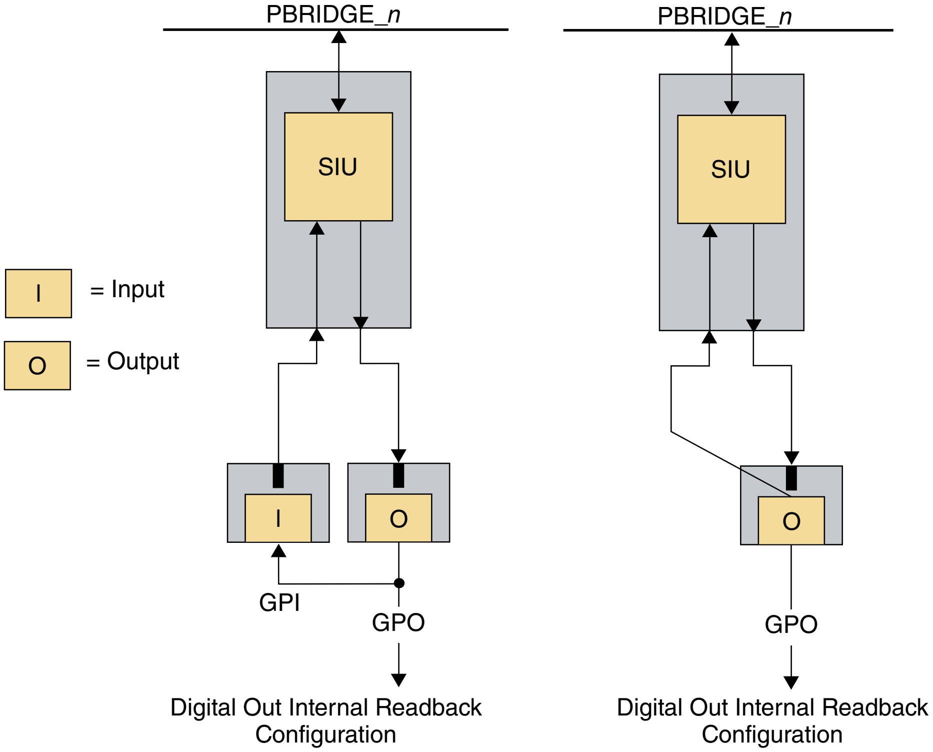

PBRIDGE_n
PBRIDGE_n
I
= Input
O
= Output
SIU
SIU
I
O
O
GPI
GPO
GPO
Digital Out Internal Readback Configuration
Digital Out Internal Readback Configuration

Figure 5-4. Single Write Digital Output With Read-Back

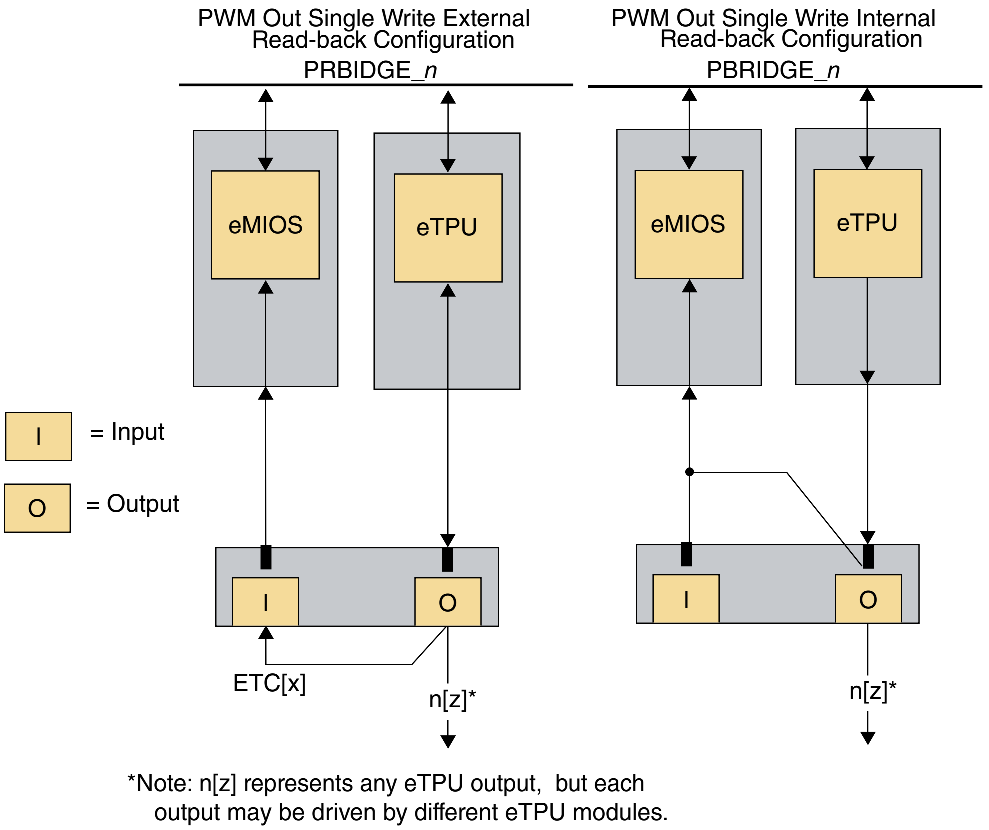

PWM Out Single Write External Read-back Configuration
PRBIDGE_n
eMIOS
eTPU
PWM Out Single Write Internal Read-back Configuration
PBRIDGE_n
eMIOS
eTPU
I
= Input
O
= Output
I
O
ETC[x]
n[z]*
I
O
n[z]*
*Note: n[z] represents any eTPU output, but each output may be driven by different eTPU modules.

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

5.3.2.1.2 Double Write Digital Output

- Double Write Digital Output

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

- Double Write PWM Output
  - The hardware elements used to perform a double write PWM output are:
    - eMIOS_0 and eMIOS_1
    - eTPU_A and eTPU_B

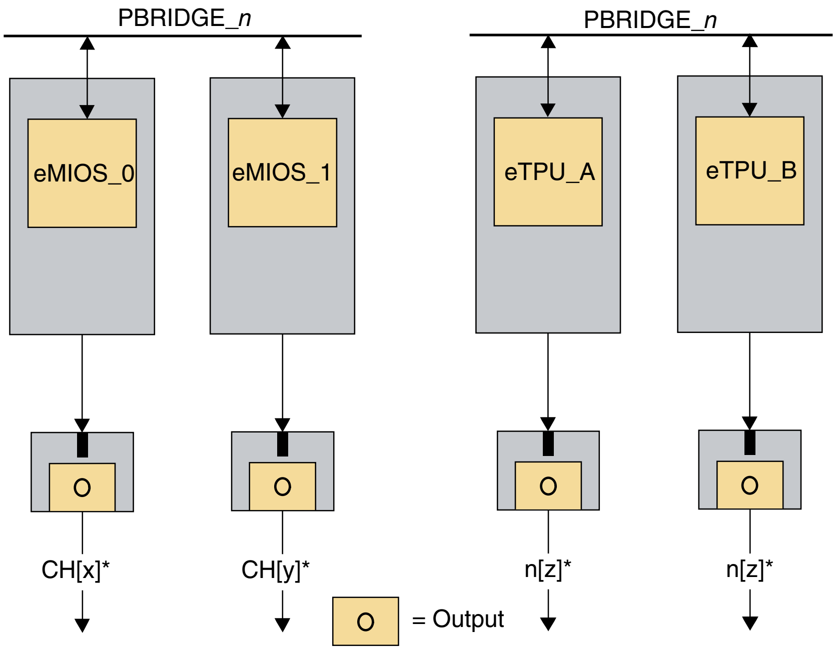


Figure 5-6. Double Write PWM Output

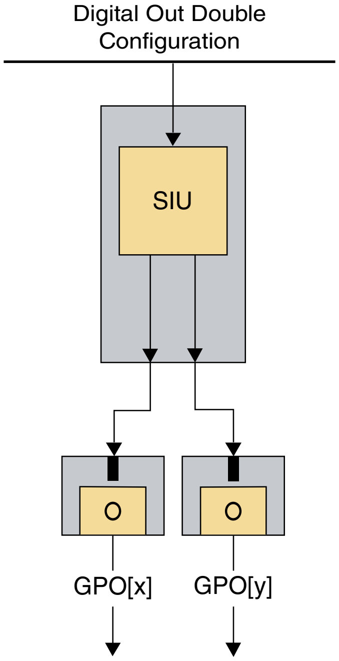


Figure 5-7. Double Write Digital Output

5.3.2.2 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 5-4 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 5-4. Digital outputs software tests

Function	Test	Frequency
Single Write Digital Outputs With Read Back	SIU_SWTEST_REGCRC	Once after programming
	GPOERB_SWTEST_CMP	Once per FTTI
	GPOIRB_SWTEST_CMP	Once per FTTI
Double Write Digital Outputs	SIU_SWTEST_REGCRC	Once after programming
	GPODW_SWAPP_WRITE	Once per FTTI
Single Write PWM Outputs With Read Back	SIU_SWTEST_REGCRC	Once after programming
	EMIOS0_SWTEST_REGCRC[1]	Once after programming
	EMIOS1_SWTEST_REGCRC1	Once after programming
	eTPU_A_SWTEST_REGCRC[2]	Once after programming
	eTPU_B_SWTEST_REGCRC2	Once after programming
	PWMRB_SWTEST_CMP	Once per FTTI
Double Write PWM Outputs	SIU_SWTEST_REGCRC	Once after programming[3]
	EMIOS0_SWTEST_REGCRC1	Once after programming
	EMIOS1_SWTEST_REGCRC1	Once after programming
	eTPU_A_SWTEST_REGCRC2	Once after programming
	eTPU_A_SWTEST_REGCRC2	Once after programming
	PWMDW_SWAPP_WRITE	Once per FTTI

[1] This test is required only if the eMIOS channels are used for the safety function.

[2] This test is required only if the eTPU channels are used for the safety function.

[3] If a change in a single SIU configuration register is capable of affecting both the output and the read-back paths, then SIU_SWTEST_REGCRC may be executed every FTTI. In all other cases configuration errors are covered by the software comparison.

5.3.2.2.1 Single Write Digital Outputs With Read-Back

The SIU hardware element is used to perform a Single Write Digital Output With Read-Back (see Figure 5-4).

Rationale: To determine whether written data is matches the expected data.

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

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

- External read back – SIU 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 – SIU 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 SIU_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 external read back or GPOIRB_SWTEST_CMP for internal read back.

5.3.2.2.1.1 Implementation details

The SIU 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.

5.3.2.2.1.2 SIU_SWTEST_REGCRC

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

5.3.2.2.1.3 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 verify that the read data equals the written data.

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

5.3.2.2.1.4 GPOIRB_SWTEST_CMP

Rationale: To verify that 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.

5.3.2.2.2 Double Write Digital Outputs

The SIU 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 SIU is configured by application software to correctly define the configuration of the outputs used. The software performs a double write.

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

5.3.2.2.2.1 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.

5.3.2.2.2.2 GPODW_SWAPP_WRITE

Rationale: To prevent SPFs in the SIU from influencing 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 (SIU_MPGPDO*n*) can be used. The two GPIOs used may be in the same SIU_MPGPDO*n* register.

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

When using a single write (atomic) instruction to SIU_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 SIU IPS interface can affect 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.

5.3.2.2.3 Single Write PWM Outputs With Read-Back

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

- eMIOS_0 – eTPU_A
- eMIOS_0 – eTPU_B
- eMIOS_1 – eTPU_A
- eMIOS_1 – eTPU_C

These units shall be configured to implement one PWM output channel and (via internal read-back) the eMIOS input PWM channel. The SIU 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 test the SIU configuration by implementing the SIU_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 eMIOS is used to read-back the output buffer of the same pad used by the eTPU output. Consult the I/O Signal Table for valid pairs of EMIOS and eTPU channels present on the same pad.

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

- ETPU_A_SWTEST_REGCRC
- ETPU_B_SWTEST_REGCRC
- ETPU_C_SWTEST_REGCRC

- EMIOS0_SWTEST_REGCRC
- EMIOS1_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 compares the written value via the read-back (PWMRB_SWTEST_CMP).

5.3.2.2.3.1 Implementation details

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

- eMIOS_0 channels
- eMIOS_1 channels
- eTPU_A channels
- eTPU_B channels
- eTPU_C channels

5.3.2.2.3.2 ETPUx_SWTEST_REGCRC and EMIOSx_SWTEST_REGCRC

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

5.3.2.2.3.3 PWMRB_SWTEST_CMP

This test compares the PWM read back provided by a single channel of the eMIOS_0 (eMIOS_1) with the expected values that have been written to the eTPU_A (eTPU_B, eTPU_C) output channel.

For this example, eTPU_A is used to generate a PWM output and eMIOS_0 is used to read back and verify the output. Another combination could be used if required in an application.

5.3.2.2.4 Double Write PWM Outputs

Rationale: The hardware elements eMIOS_0, eMIOS_1, eTPU_A, eTPU_B, and eTPU_C are used to perform a double Write PWM Output.

Implementation hint: These units are configured to implement two independent PWM channels. The SIU 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 SIU.

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 SIU configuration by implementing the SIU_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:

- EMIOS0_SWTEST_REGCRC (for eMIOS)
- EMIOS1_SWTEST_REGCRC (for eMIOS)
- ETPU_A_SWTEST_REGCRC (for eTPU)
- ETPU_B_SWTEST_REGCRC (for eTPU)
- ETPU_C_SWTEST_REGCRC (for eTPU)

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 eMIOS (eMIOS_0, eMIOS_1) or one of the three eTPUs (eTPU_A, eTPU_B,eTPU_C).

5.3.2.2.4.1 Implementation details

The following hardware elements are used for the safety function:

- eMIOS_0 channels
- eMIOS_1 channels
- eTPU_A channels

- eTPU_B channels
- eTPU_C channels

5.3.2.2.4.2 SIU_SWTEST_REGCRC

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

5.3.2.2.4.3 PWMDW_SWAPP_WRITE

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

5.4 Communications

5.4.1 Redundant communication

Portions of the integrated DSPI and eSCI communication controllers do not independently provide the functional safety integrity IEC 61508 series and ISO 26262 require for high functional safety integrity targets. As these communication protocols often deal with low complex slave communication nodes, higher level functional safety protocols as described in Fault-tolerant communication protocol may not be feasible. Therefore, appropriate communication channel redundancy may be required. Multiple instances of communication controllers may be used to build up a single fault robust communication link.

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

- Synchronous Serial Communication Controller (DSPI)
- eSCI Communication Controller

DSPI and eSCI do not have special functional safety mechanisms other than what is included in their protocol specifications. The system level communication architecture needs to provide the functional safety mechanisms on the interface of the modules to meet the assumptions.

5.4.2 Fault-tolerant communication protocol

Parts of the integrated eSCI, MCAN and FlexCAN communication channels do not independently provide the functional safety integrity IEC 61508 series and ISO 26262 require for high functional safety integrity targets.

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

- FlexCAN Communication Controller
- MCAN Communication Controller
- Enhanced Serial Communication Interface (eSCI)

FlexCAN, MCAN, and eSCI 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 assumptions.

Typical 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
- 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 modules 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 eSCI, only a simplified functional safety protocol layer may be required.

5.5 Additional configuration information

5.5.1 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 too 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.

5.5.1.1 Initial checks and configurations

Assumption under certain conditions: [SM_103] When stack underflow and stack overflow due to systematic faults within the application software endanger 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: The Data Address Compare 1 and 2 (DAC1, DAC2) resources maybe used for incremental stack overflow or stack underflow detection when not used as a hardware or software debug resource. Stack limit checking is available regardless of EDM or IDM mode, and when resources used for stack limit checking are owned by software, will utilize a DSI or machine check exception.

A 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 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 only enables 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.

5.5.2 MPC5777C configuration

Assumption: [SM_240] It is required that application software verifies that the initialization of the MPC5777C is correct before activating the safety-relevant functionality. [end]

After startup, the application software shall 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
- ERROR*n* signal check[4]

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

Assumption: [SM_241] It is required that application software checks the configuration of the SSCM once after boot.[end]

Assumption: [SM_280] Decorated storage transactions on the SRAM are limited to read-modify-write at least for the safety core. Notice that test and set (for example, individual bit manipulations) are still allowed as single bit read-modify-write (but no AND/OR operation is possible). [end]

[4] Required for single FCCU signal usage only.

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

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

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

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

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

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

Rationale: To reduce the risk accidental writes to configuration registers affecting the execution of the MPC5777C's safety function or disable the safety mechanism due to their change.

Recommendation: All configuration registers, and registers that are not modified during application execution, should be protected using Peripheral Access Control.

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

Chapter 6
Failure rates and FMEDA

6.1 Failure rates

In order to analyze and quantify the effectiveness of the MPC5777C 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 MPC5777C. The following methods for deriving the base failure rates of the MPC5777C 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 MPC5777C 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 MPC5777C enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request.

6.2.1 Module classification

For calculating the safety metrics for ISO 26262 (Single-Point Failure Metric (SPFM), Latent Failure Metric (LFM) and Probabilistic Metric for random Hardware Failures (PMHF)) and for IEC 61508 (Safe Failure Fraction (SFF) and β_{IC} factor) the modules of the MPC5777C 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 MPC5777C can be found in the attached "MPC5777C 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 MPC5777C 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 master and checker core together with related logic are separated in this way as well as the redundantly available peripheral modules.

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 MPC5777C and is capable of detecting this CCF, and will switch the system to a Safe state$_{\text{system}}$ within the FTTI.

The MPC5777C device satisfies the standard AECQ100 for latch-up immunity.

7.1.2.2 Environmental conditions

7.1.2.2.1 Temperature

The MPC5777C was designed to work within a maximum operational temperature profile (see the *MPC5777C 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 (14 x CMU). Major failures in the clock system are also detected by the SWT (Software Watchdog Timer).

7.1.2.3.2 Power supply

To cover dependent failures caused by issues with the power supplies, supervision modules are implemented (see Power Management Controller (PMC)). Some dependent failures (for example, loss of power supply) are detected since software will no longer be able to trigger the external watchdog (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 MPC5777C 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 *MPC5777C 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 *MPC5777C Data Sheet*) together with the ball number information of the packages as seen in the *MPC5777C 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 MPC5777C 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 assocaiated 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 MPC5777C is not able to signal an internal failure via its ERROR*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 MPC5777C 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 MPC5777C. As soon as there are no correct communications, the external watchdog function switches the system to Safe state$_{\text{system}}$. Thus, either the MPC5777C or external watchdog function can transition the system to Safe state$_{\text{system}}$. The external watchdog function is required to be sufficiently independent of the MPC5777C (for example, regarding clock generation, power supply, and so on).

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

7.1.4 β_{IC} considerations

During the development of the MPC5777C, 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																															
	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.

Appendix A
Release Notes for Revision 2.1

A.1 General changes in this document

- Rev. 2.1 changes the document from Freescale to NXP branding:
    - All other content in Rev. 2.1 is the same as in Rev. 2.
    - All other changes in this appendix are relative to Rev. 1.
- Editorial changes and improvements throughout the document.

A.2 Preface changes

- Editorial updates throughout the chapter.

A.3 MCU safety context changes

- Removed section 'Module classification' since it is duplicated in chapter 6.
- In the Faults section:
    - In the paragraph describing the occurrence of FTTI/L-FTTI, changed "by test software (for example, BIST after power-up)" to "and potential faults are typically detected by safety mechanisms which are executed during system testing at startup".
- In the Dependent failures section:
    - Renamed the "Failures" section to "Dependent failures".
- In the Latent-fault tolerant time interval for latent faults section:
    - Removed the statement "Within this time frame, the Safety Element out of Context (SEooC) shall be considered unsafe."

- In MCU safety functions :
    - Editorial update.
- In Correct operation :
    - Changed "Software Execution Function" to "MCU Safety Function".

- In the Faults section:
    - Editorial changes to the "Latent Fault" bullet for better clarity.

- In the Safety integrity level section:
    - Editorial updates (changed "ISO26262" to "ISO 26262", and "IEC61508" to "IEC 61508").

- In Faults and failures :

  - Updated the explanation of random hardware fault reduction and detection for improved clarity.

- In Latent-fault tolerant time interval for latent faults :
  - Editorial change.
  - Added Assumption SM_212 and Rationale.

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

A.4 Functional Safety Concept changes

- In the General functional safety concept section:
  - Updated the block diagram.
  - Removed the assumption: "Before starting safety-relevant operations, the application software shall check that the checker core is enabled and configure the FCCU to react to LSM being disabled."
  - Removed the rationale: "As LSM is transparent to the system level (for example, to application software), specific requirements must be fulfilled to improve functional safety integrity in case the device is intended to operate in LSM."

A.5 Hardware requirements changes

- 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".
  - Removed the recommendation to disable the system when an overvoltage occurs.

- In Hardware requirements on system level :
  - Updated the introductory text related to the application schematic figure.
  - Updated the "Functional safety related connection to external circuitry" application schematic figure.
- In External Watchdog (EXWD) :
  - Changed "FlexCAN" and "MCAN" to simply "CAN".

- In the Hardware requirements on system level section:
  - Removed the Notes.

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

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

- In the PowerSBC section:

- Updated the "Functional safety application with PowerSBC" figure.

- In the Hardware requirements on system level section:
  - Corrected instance of FCCU_F[n] to ERROR*n*.
- In the PowerSBC section:
  - Changed the generated voltage descriptions to "5 V (Vaux), 3.3V (Vcca), and 1.25V (Vcore) to supply the MPC5777C"

- In the Both FCCU signals connected to separate device section:
  - Moved "Rationale" to be after "Assumption".
  - Removed the pin behavior table.
- In the High impedance outputs section:
  - Removed the redundant introductory paragraph.
  - Removed the "Implementation hint" heading.

A.6 Software Requirements changes

- In the Software requirements on system level section:
  - Added the device number MPC5777C, which was missing in three places.
  - Editorial changes.

- In the 1.25 V supply supervision section:
  - Editorial change.
  - In the Implementation hint, removed the content related to external ballast transistors.
- In the Test mode section:
  - Removed the recommendation.

- In the Mulitplexed serial communicaton protocol controllers section:
  - Added new paragraph, "FlexCAN modules are redundantly available..."
- In the FCCU Runtime checks section:
  - Changed "resetting functional and destructive reset counters" to "resetting the reset counters".
- In the Power Management Controller (PMC) section:
  - Changed "a destructive reset" to "a reset".
  - Updated the "PMC monitored supplies" table.
- In the 1.25 V supply supervision section:
  - Changed "a destructive reset" to "a reset".
- In the 3.3 V supply supervision section:
  - Changed "a destructive reset" to "a reset".
  - Removed SM_191 and the associated text.
- In the Built-In Hardware Self-Tests (BIST) section:
  - Changed "after a reset (destructive reset or external reset)" to "after a reset".
  - Changed "a destructive or external reset" to "a power-on, FOSU, or external reset".
  - Removed the redundant paragraphs "Boot time test..." and "All tests may..."
  - Editorial change.
- In the SD-ADC Initial checks and configurations section:
  - Changed occurrences of "destructive reset" to "power-on reset".

- In the Self Test Control Unit (STCU2) section:
  - Removed the statement that faults are signaled to the FOSU.
- In the Software Watchdog Timer (SWT) section:
  - Added SM_067.
  - Added SM_102.
  - Changed "Assumption: It is expected that application software uses the SWT..." to "In general, it is expected that application sofware uses the SWT..."
  - Removed the following paragraph and its bullets: "Assumption: These requirements apply to the SWT for safety-relevant applications:".
- In the RCCU Initial checks and configurations section:
  - Added missing device number.
  - Added SM_033.

- In the CRC Cyclic Redundancy Checker Unit section:
  - Added missing device number.
- In the PLLDIG Initial checks and configurations section:
  - Changed "system level functional safety integrity measure respective functional safety-relevant modules shall be clocked..." to "system level functional safety integrity measure or functional safety-relevant modules, or both, they shall be clocked..."
- In the Power Management Controller (PMC) section:
  - Made improvements to the description of the monitored voltages behavior.
  - Added SM_204.
  - In the paragraph "Apart from self test...", added "and ADC monitoring of the bandgap reference voltage".
- In the Built-In Hardware Self-Tests (BIST) section:
  - Removed software based self-test (SBST) from the list of checking for latent faults.
  - Added SM_109.
- In the STM Runtime checks section:
  - Removed the word "Assumption" from the start of the first paragraph.
- In the EEPROM section:
  - Added this new section.
- In the EEPROM Initial checks and configurations section:
  - Changed the first Implementation hint to be SM_112.
- In the EEPROM Runtime checks section:
  - Updated SM_219.
- In the Error reporting path tests section:
  - Removed the Assumption (SM_224).
  - Removed the Rationale.
- In the Glitch filter section:
  - Added missing device number.
- In the Crossbar Switch (XBAR) section:
  - In the first sentence, removed the concurrent transactions master and slave examples.
  - In Assumption SM_227, changed "SFSMo or SuMos" to "safety-related modules".
  - Removed the unnumbered (second) Assumption.
- In the eQADC Enhanced Queued Analog to Digital Converter (eQADC) section:
  - Added this new section.
- In the Initial checks and configurations section:
  - Added this new section.
- In the I/O functions section:
  - Removed the statement regarding enablement of integrity measures on the hardware level and inclusion of I/O bridges and the crossbar switch.

A.7 Failure Rates and FMEDA changes

- In the FMEDA section:
  - Changed "Dynamic FMEDA" to "FMEDA".
  - Editorial change.

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

A.8 Dependent Failures changes

- 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 : • 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 MPC5777C Reference Manual".
• Editorial updates throughout chapter.
• In Dependent failure avoidance on system level : • Removed the Recommendation as it was redundant. • In External timeout function : • Removed the sentence that immediately followed the note, as it was redundant.
• In Physical isolation : • In the second paragraph, changed the word "fatal" to "catastrophic".
• In Physical isolation : • Added the following: "The master and checker core together with related logic are separated in this way as well as the redundantly available peripheral modules."

A.9 Additional Information changes

• In ECC checkbit/syndrome coding scheme : • Changed "E2E ECC" to "e2eECC" throughout the section.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Document Number MPC5777CSM
Revision 2.1, 02/2017