Safety Manual for MPC5746R

Devices Supported: MPC574xR, MPC574xF

Document Number: MPC5746RSM
Rev. 2.1, 07/2017

Contents

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

Chapter 1
Preface

Chapter 2
General information

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

Chapter 9
Acronyms and abbreviations

Chapter 1
Preface

1.1 Preface

[SM_001] This document discusses requirements and assumptions for the use of the MPC5746R Microcontroller Unit (MCU) in ISO 26262 safety-related applications. [end]

This document is intended to support system and software engineers using the MPC5746R features as well as achieving additional diagnostic coverage by software measures.

This document takes the following into consideration:

- The system containing the MPC5746R MCU
- The "Safety Element out of Context" section in the "Road vehicles - Functional safety - Part 10: Guideline [ISO 26262-10]" standard
- Certain assumptions about the system's functional safety needs based on that standard

and determines whether a measure is mandatory based on these factors.

Several measures are prescribed as safety requirements whereby the measure described was assumed to be in place when analyzing the functional safety of the MPC5746R. In this sense, requirements in the Safety Manual (SM) are driven by assumptions concerning the functional safety of the system that will integrate the MPC5746R.

- **Assumption:** An assumption that is relevant for functional safety in the specific application under consideration (condition of use). It is assumed that the user fulfills an assumption in his design.
- **Assumption under certain conditions:** An assumption that is relevant under certain conditions. It is assumed that the user fulfills an assumption in his design, if the associated condition is met.

Example: **Assumption:** It is assumed that the recommended operating conditions given in the MPC5746R Data Sheet are maintained.

Example: **Assumption under certain conditions:** If an output in high-impedance is not considered safe at the system level, it is assumed that countermeasures are in place to bring the safety-critical outputs to their Safe state.

NOTE

Assumptions (or assumptions under certain conditions) are marked by a tag of the form “SM_nnn” at the beginning of the assumption, and are terminated with an “end”. Both of these tags are enclosed within square brackets for easy recognition. These tags could be used to import the assumptions into safety traceability management tools.

If a specific safety manual assumption is not fulfilled, an alternate implementation has to be shown that it is at least similarly effective concerning the functional safety requirement in question (for example, provides same coverage, reduces the likelihood of Common Mode Failure (CMF) similarly well, and so on). If the alternative implementation is shown to be not as effective, the estimation of an increased failure rate (λ_{SPF}, λ_{RF}, λ_{MPF},…) and reduced metrics (SPFM: Single-Point Fault Metrics, LFM: Latent Fault Metric) due to the deviation must be specified.

This document also contains guidelines on how to configure and operate the MPC5746R for functional safety relevant applications requiring high functional safety integrity levels. 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 user has option to implement the recommendation.
- **Rationale:** The motivation for a specific assumption and/or recommendation.
- **Implementation hint:** An implementation hint gives specific hints on the implementation of an assumption and/or recommendation on the MPC5746R. The user has the option of whether to apply the implementation hint.

These guidelines are considered to be useful approaches for the specific topics under discussion. The user will need to use discretion in deciding whether these measures are appropriate for their applications.

[SM_002]This document is valid only under the assumption that the MPC5746R is used in functional safety applications requiring a fail-silent or a fail-indicate MCU.[end] A fail-operational mode of the MPC5746R is not described.

This document targets high functional safety integrity levels. For functional safety goals that do not require high functional safety integrity levels, system integrators will need to tailor the requirements for their specific application.

It is assumed that the user of this document is generally familiar with the MPC5746R and ISO 26262 standards.

Assumption: [SM_003] All relevant hardware safety mechanisms are enabled and configured correctly when using any of the information in this document. [end]

1.1.1 Related documents

The following lists the documentation referenced throughout the Safety Manual:

- ISO 26262: ISO 26262 *Road vehicles – Functional safety*, November 2011
- MPC5746RRM, *MPC5746R Reference Manual*
- MPC5746R, *MPC5746R Data Sheet*
- FMEDA
- *Addressing the Challenges of Functional Safety in the Automotive and Industrial Markets*, White Paper, October 2011
- See Freescale Semiconductor's SafeAssure Functional Safety Program for additional information.

1.1.2 Vocabulary

The vocabulary defined in ISO 26262-1 applies to this document.

Specifically, the following terms apply:

- **System:** functional safety related system that implements the required functional safety goals necessary to achieve, or maintain, a Safe state$_{system}$ for the equipment under control (control system). It is also intended to achieve, on its own, or with other functional safety related systems and other risk reduction measures, the functional safety integrity for the required safety functions.
- **System integrator:** person who is responsible for the system integration.
- **Element:** part of a subsystem comprising a single component, or any group of components (for example, hardware, software, hardware parts, software units), that perform one or more safety functions (functional safety requirements).
- **Trip time:** the maximum time of operation of the MPC5746R without switching to power down state.

Chapter 2
General information

2.1 General Information

The MPC5746R 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).

2.1.1 Assumed conditions of operation

Assumption:[SM_034] This document is only valid if the operating conditions given in the *MPC5746R Data Sheet* are maintained.[end]

Assumption:[SM_037] The MPC5746R shall be handled according to JEDEC standards J-STD-020 and J-STD-033. [end]

Assumption:[SM_035] This document is only valid if the recommended production conditions given in the specific MPC5746R quality agreement are maintained.[end]

Assumption: It is assumed that all field failures of the MPC5746R are reported to the silicon supplier.

Rationale: To cover ISO 26262-7 (6.5.4) and ISO 26262-7 (6.4.2.1).

Assumption:[SM_036] The latest MPC5746R errata shall be taken into account during system design, implementation, and maintenance. For a functional safety related device such as the MPC5746R, this also concerns functional safety related activities such as the development of the system functional safety concept.[end]

2.1.2 Safety Function

2.1.2.1 MCU safety functions

Given the application independent nature of the MPC5746R, no specific safety function can be specified. Therefore, during the SEooC development of the MPC5746R, 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 MPC5746R flash memory, buffer these within instruction cache, execute instructions, read data from the MPC5746R System SRAM or flash memory, buffer these in data cache, process data and write result data into MPC5746R System SRAM. **Functional safety of the Software Execution Function is primarily achieved by safety mechanisms integrated on the MPC5746R**.

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

- **Input / Output Functions (Application dependent)**: Input / Output functions of the MPC5746R have a high application dependency. **Functional safety will be primarily achieved by system level safety measures**.
- **Debug Functions**: It is assumed that debug functions are **Not Safety Related**.

2.1.2.2 Correct operation

Correct operation of the MPC5746R 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.

2.1.3 Mission Profile

Lifetime for the MPC5746R is 20 years which is equivalent to 20,000 hrs of active operation. The assumed mission profile is:

- Lifetime (T_{life}): 20 years
- Total operating hours: 20,000 hrs
- Trip time (driving cycle, T_{trip}): 12 hrs
  - [SM_004] Trip time is the maximum time of operation of the MPC5746R without a power-on reset. [end]
- Fault-Tolerant Time Interval (FTTI): 10 ms.
  - FTTI is the time the control system will not transition to a hazardous state, despite failure of the MPC5746R.

Note

This is a conservative estimate since the actual numbers will depend on the MPC5746R application (See Single-point Fault Tolerant Time Interval and Latent-FTTI for latent faults for exact calculation instructions).

2.1.4 Functional safety – ISO 26262 compliance

[SM_006] The MPC5746R was developed in accordance with ISO 26262 as a Safety Element out of Context (SEooC). [end]

The MPC5746R is suitable to be used in safety-relevant applications including systems that are classified as ISO 26262 ASIL A, ASIL B, ASIL C or ASIL D.

[SM_007] The development process of the MPC5746R fulfills ASIL D requirements of ISO 26262. [end]

2.1.5 Safety goals

The safety goals of the MPC5746R are defined as follows:

- [SM_008] The **primary safety goal** is that the MPC5746R does not leave its Safe $state_{MCU}$ for intervals equal to or longer than the FTTI (10 ms) unless configured by the application software to do so. [end]
- [SM_009] The **secondary safety goal** is that the MPC5746R, or the software running on the MPC5746R, shall be able to detect any permanently unavailable safety mechanism that is necessary to reach the primary safety goal. This shall be done at least once per driving cycle (12 hrs). [end]

2.1.6 Safe state

Safe state of the system is named Safe state$_{system}$, whereas a Safe state of the MPC5746R is named Safe state$_{MCU}$. The Safe state$_{system}$ of a 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$_{system}$ may be the intended operating mode, or a mode where the system has been disabled.

[SM_010] The safety goals are achieved by transitioning or holding the MPC5746R in the following safe states (see the following figure): [end]

- [SM_011] Completely unpowered (see Figure 2-1, diagram (d)) [end]
- [SM_012] In reset (see Figure 2-1, diagram (c)) [end]
- [SM_013] Operating correctly (see Figure 2-1, diagram (a), and Correct operation) [end]
- [SM_014] Explicitly indicating an internal error (indication on ERROR[0:1], Figure 2-1, diagram (b)) [end]

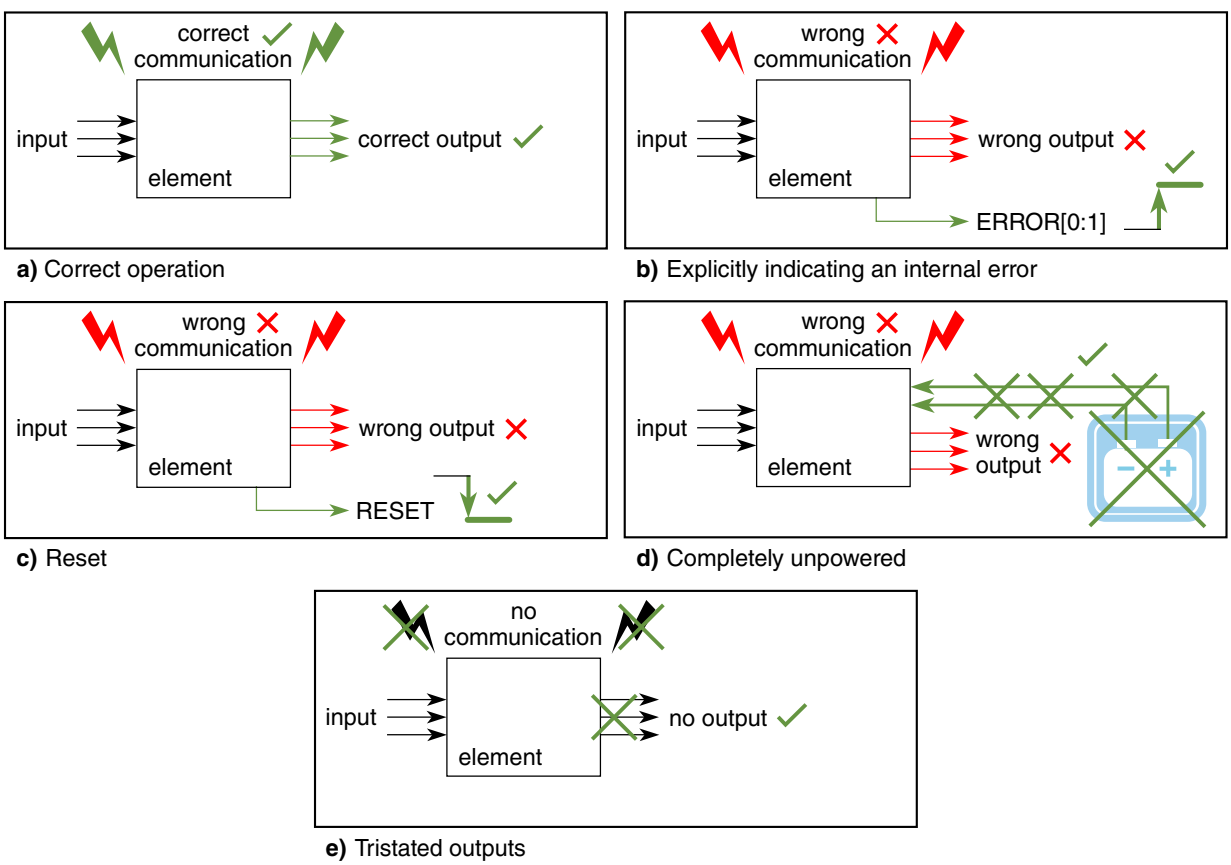


Figure 2-1. Safe state$_{MCU}$ of MPC5746R

Assumption:[SM_015] The system transitions itself to a Safe $state_{system}$ when the MPC5746R explicitly indicates an internal error (as shown on ERROR[0] or ERROR[1]). [end]

Assumption: [SM_016] The system transitions itself to a Safe $state_{system}$ when the MPC5746R is in reset state. [end]

Assumption: [SM_017] The system transitions itself to a Safe $state_{system}$ when the MPC5746R is unpowered. [end]

Assumption: [SM_018] The system transitions itself to a Safe $state_{system}$ when the MPC5746R has no active output (for example, tristate). [end]

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 (See Consecutive resets for details).

Assumption:[SM_019] The application must identify, and signal, continuous switching between reset and standard operating mode as a failure condition. [end]

[SM_158] If the MPC5746R signals an internal failure via its error out signals (ERROR[0:1]), the surrounding subsystem shall no longer use the MPC5746R outputs for safety functions since these signals can no longer be considered reliable. If an error is indicated, the system must be able to remain in a Safe $state_{system}$ without any additional action by the MPC5746R. Depending on the configuration, the system may disable, or reset, the MPC5746R as a reaction to the error signal. [end]

2.1.7 Single-point Fault Tolerant Time Interval

The single-point Fault Tolerant Time Interval (FTTI) is the time span between recognition of a failure, that has the potential to cause a hazardous event, and the time by which an action has to be completed to prevent the occurrence of the hazardous event. It is used to define the sum of worst case fault indication time and the time for execution of a corresponding countermeasure (reaction). The figure below shows the FTTI for a single-point fault in a system.

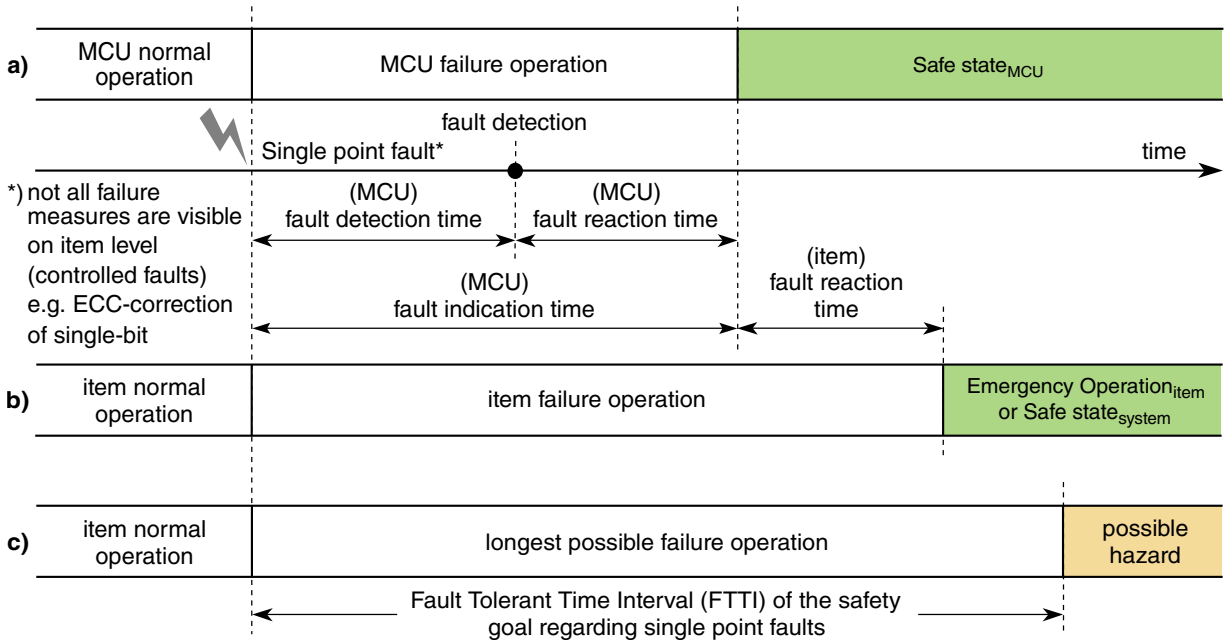


Figure 2-2. Fault tolerant time interval for single point faults

Fault indication time is the time from the occurrence of a fault to when the MPC5746R is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the MPC5746R to a high impedance state, or by assertion of reset).

Fault detection time has five components, two of which are influenced by configuration settings:

- recognition + software cycle + software execution + internal processing + external indication

Each component of fault indication time is described as follows:

- **Diagnostic test interval** is the interval between online tests (for example, software based self test) that allows for detection of 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.
- **Fault detection time** is the maximum detection time of all involved functional safety mechanisms. The three mechanisms with the longest time are:
    - ADC[1] recognition time is a very demanding hardware test. The self-test requires the ADC conversion to complete a full test. A single full test takes at least 70 µs.

1. ADC recognition time is relevant only if ADC is used by the safety function.

- 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** is the maximum of the reaction time of all involved functional safety mechanisms.
  - **Internal processing time** to communicate the fault to the Fault Collection and Control Unit (FCCU). This 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 MPC5746R. This time depends on the indication protocol configured in the FCCU:
    - Dual Rail protocol and time switching protocol:
      - **FCCU configured as "fast switching mode"**: indication delay is a maximum of 64 µs. As soon as the FCCU receives a fault signal, it reports the failure to the system.
      - **FCCU configured as "slow switching mode"**: an indication delay could occur. The maximum delay is equal to the duration of the semiperiod of the error out (FCCU_Fn) 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).

[SM_020] The sum of the MPC5746R fault indication time and system fault reaction time shall be less than the FTTI of the functional safety goal. [end]

2.1.8 Latent-FTTI 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 with other latent faults and give rise to a hazardous multiple-point event, and the time by which counteraction has to be completed to prevent the hazardous event from occurring. It is used to define the sum of respective worst case

fault indication time and the time for execution of the corresponding countermeasure. Within this time frame, the safety element out of context (SEooC) shall be considered unsafe. The figure below shows the L-FTTI for multiple-point faults in a system.

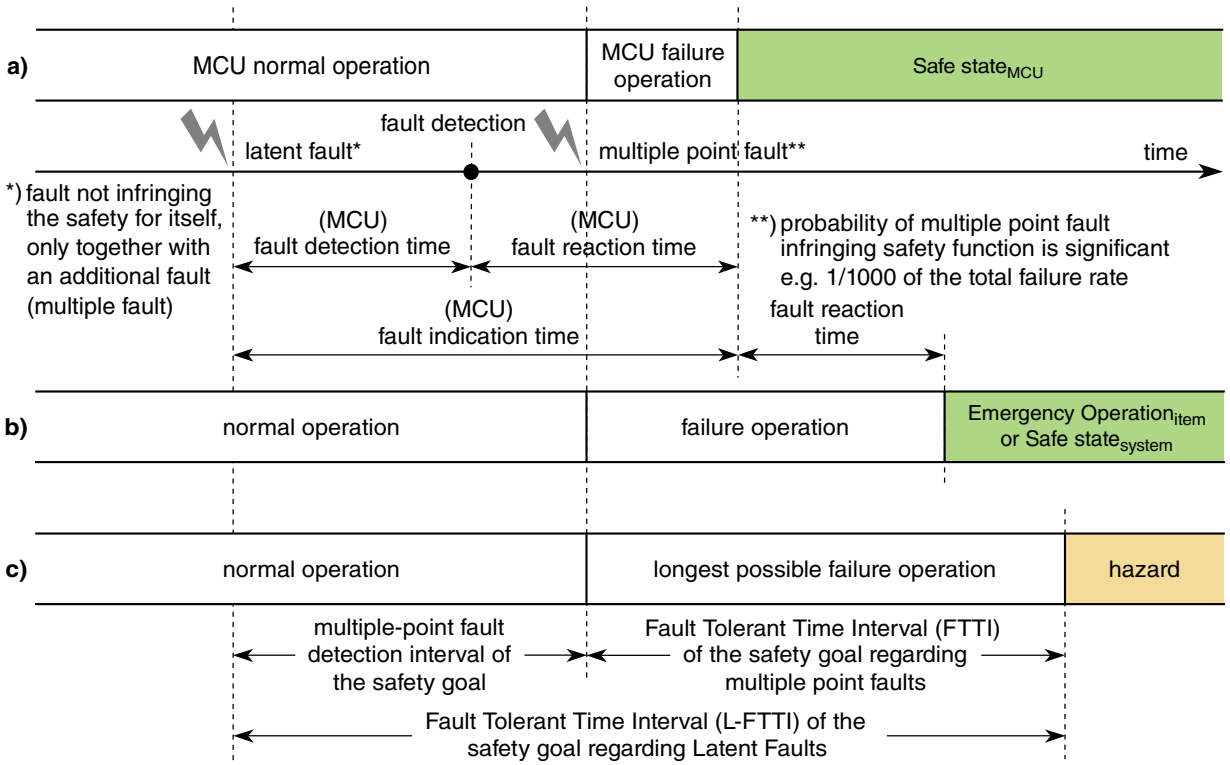


Figure 2-3. Fault Tolerant Time Interval for latent faults

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 the error out signals (ERROR[*n*]), forcing outputs of the MPC5746R to a high impedance state (Safe Mode) or by assertion of reset.

Fault detection time has five components, two of which are influenced by configuration settings:

- recognition + internal processing + external indication + software cycle + software execution

Each component of fault indication time is described as follows:

- **Diagnostic test interval:** the interval between online tests (for example, software based self-test) to detect faults in a functional safety related system that has a specified diagnostic coverage. This time depends closely on the system level implementation (for example, software).
    - **Software cycle time** of software based functional safety mechanisms. This time depends closely on the software implementation.

- **Fault detection time** is the maximum of the detection time of all involved functional safety mechanisms. The mechanisms with the longest times are:
  - Single-bit corrected permanent hardware SRAM fault – This fault is only controlled, and is not reported to the operator of the system. Therefore, it is a latent triple fault scenario, as ECC has a reduced capability to detect triple bit faults. The L-FTTI is in the range of 1×10^9 $h^{-1} \approx 2 \times 10^5$ years for a permanent single-bit fault, or ≈ 20 years continuous operation for 10,000 faults.
  - **Software execution time** of software based functional safety mechanisms. This time depends closely on the software implementation.
- **Fault reaction time** contains the following:
  - **Internal processing time** to communicate the fault to the Redundancy Checker Control Unit (RCCU), and is a maximum of 10 IRCOSC clock cycles (nominal frequency of 16 MHz).
  - **External indication time** to notify an observer about a failure external to the MPC5746R. This time depends on the indication protocol configured in the 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 period of the FCCU_F[*n*] signal (as configured in FCCU_CFG[FOP]). This parameter configured to be configured equal to its minimum which is 128 µs.
    - **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.

In general, Internal processing, Indication and Execution times are negligible for multiple-point failures since the L-FTTI is significantly larger than typical Processing, Indication and Execution times.

The sum of latent fault indication time and latent and multiple point fault reaction time shall 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.

2.1.9 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 MPC5746R initialization). These errors are required to be handled before the system enables 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 MPF detection interval.

Assumption: [SM_021] Boot-time failure handling shall be handled before the safety function starts. [end]

Assumption: [SM_022] Single-point and latent failure diagnostic measures shall complete operations (including fault reaction) in a time shorter than the respective FTTI, MPF detection interval. Alternatively, single-point and latent failure diagnostic measures shall complete operations (including fault reaction) before enabling system level safety function. [end]

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 MPC5746R.

2.1.10 Failure indication signaling

The FCCU offers a hardware channel to collect errors and bring the MPC5746R to a Safe $state_{MCU}$ when an internal failure is present. [SM_027]The FCCU provides two non-multiplexed error outputs (ERROR[0] and ERROR[1]) used for external failure indication. [end]

[SM_028] Different protocols for the error output pins are supported: [end]

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

[SM_029] After power-on reset, ERROR[*n*] 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. [end]

[SM_030] At least one of the ERROR[*n*] signals will be high to indicate that the MPC5746R is in the operational state. [end] [SM_031] If a two-pin bi-stable protocol with differential outputs is implemented (for example, ERROR[0] = 0 and ERROR[1] = 1 and vice-versa), the application software can configure which ERROR[*n*] signal will be high to indicate the operational state. [end] (see Error Out Monitor (ERRM) for details on requirements for connecting ERROR[*n*] to external devices).

Chapter 3
Functional safety concept

3.1 Functional safety concept

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 faults (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. Thus, measures reducing the likelihood of random hardware faults is either the detection and control of the faults during the lifetime, or reduction of failure rates. 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.

3.1.1 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 to a single-point failure. This leads directly to the violation of a safety goal. Figure 3-1 shows a SPF inside an element, which generates a wrong output.
- **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 goal. Figure 3-1 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 as safe faults and are not covered in the functional safety concept of MPC5746R.
- **Latent Fault (LF):** a MPF whose presence is not detected by a safety mechanism nor perceived by the driver. A LF is a fault which does not violate the functional safety goal(s) itself, but it leads to a dual-point or multiple-point failure when in combination with at least one additional independent fault, which then leads directly to the violation of a functional safety goal. Figure 3-1 shows a LF inside an element, which still generates a correct output.
- **Residual Fault (RF):** a portion of a fault that by itself leads to the violation of a safety goal, where that portion of the fault is not covered by a functional safety mechanism. Figure 3-1 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 safety goal. Safe faults are not covered in this document. SPFs, RFs or DPFs are not safe faults.

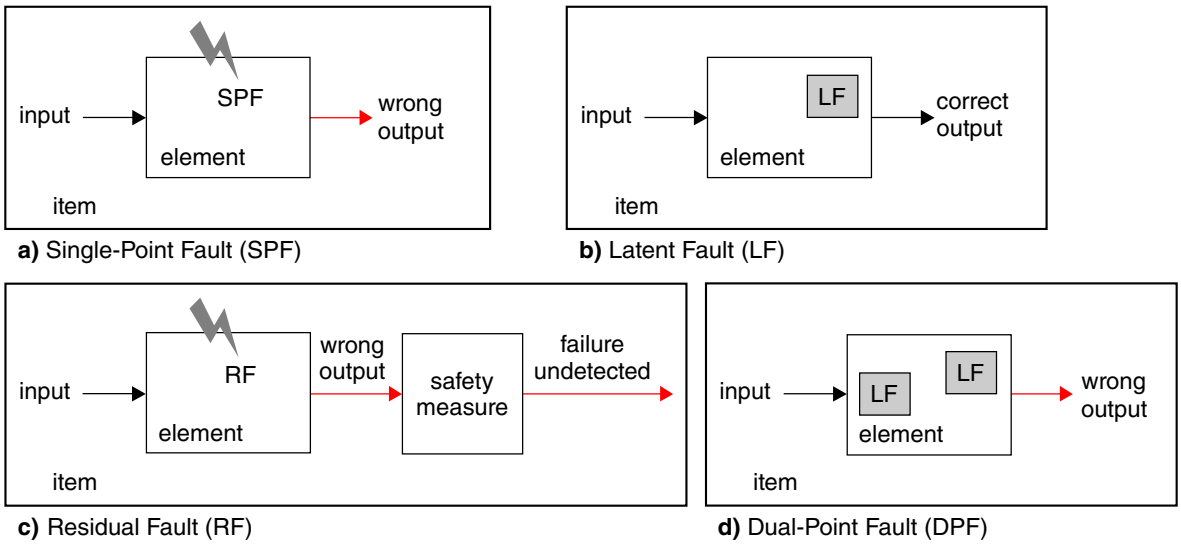


Figure 3-1. Faults

SPFs shall be detected within the FTTI. LFs (DPFs) shall be detected within the L-FTTI. In automotive applications, L-FTTI is generally accepted to occur once per typical automotive T_{trip} by the test routines (for example, BIST after power-up). Detecting DPFs once per T_{trip} reduces the accumulation time of latent faults in T_{life} of the product, to T_{trip}.

Mission Profile lists a profile with a typical trip time for automotive applications

3.1.2 Failures

- **Common Cause Failures (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.

  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). CCF causes the probability of multiple channels (N) of having a failure rate larger than $\lambda_{\text{single channel}}^{N}$ ($\lambda_{\text{redundant element}} > \lambda_{\text{single channel}}^{N}$).

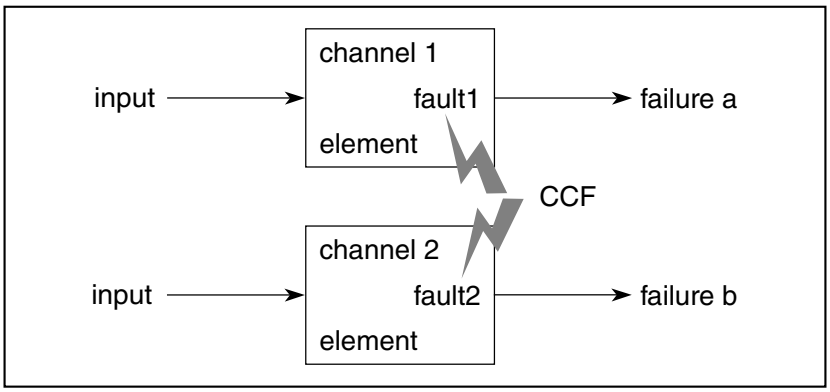


Figure 3-2. Common Cause Failures

- **Common Mode Failures (CMF):** A subset of CCF. Coincidence of random faults within two or more (not necessarily identical) elements in redundant channels leading to identical coincidental erroneous behavior with respect to the safety function. As the failures are identical, they are not detected by a comparator. Figure 3-3 shows two elements within two different, but redundant, channels, to which one single root cause leads to two different faults (fault 1, fault 2) resulting in an identical failure (failure a) in both elements and in both channels. As the identical failure occurs in both channels, the functional safety comparator mechanism does not detect the failure.

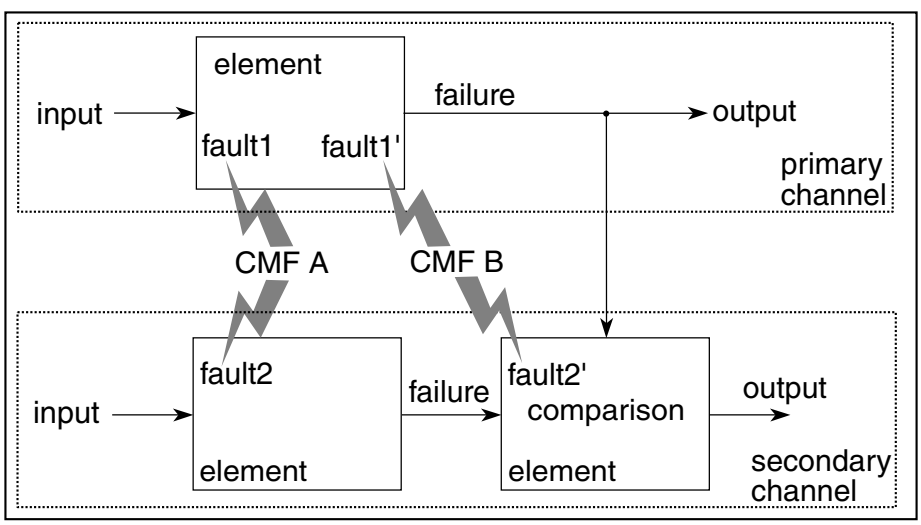


Figure 3-3. Common Mode failures

- **Cascading Failures (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 3-4 shows two elements within a single channel, to 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).

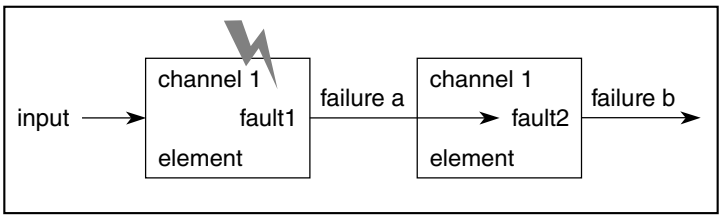


Figure 3-4. Cascading failures

3.1.3 General functional safety concept

The block diagram in the 'Introduction' of the *MPC5746R Reference Manual* shows the module interconnection.

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.
- [SM_149] Replication of processing elements are as follows: [end]
  - [SM_150] Core [end]
  - [SM_151] Cache controller [end]
  - [SM_152] Core Local Memory Controller[end]
  - [SM_153] eDMA [end]
- [SM_032] For the dual cores and their closely related periphery, functional safety is improved by a lockstep approach. [end] Any deviation in the output of the two 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
  - System RAM
  - Overlay RAM
  - D-MEM
  - I-MEM
  - FlexCAN
  - Cache and cache tags

    - eDMA Transfer Control Descriptor (TCD) RAM
    - eTPU SCM and SDM
- 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 latent failures and therefore reduce 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 MPC5746R ensures that redundant use of peripherals is protected against CMF.
- Usage of internal (and external) watchdogs or timeout measures.

[SM_154] The MPC5746R 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. [end] 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.

The MPC5746R supports only static configuration of LSM(for example, no dynamic switching between lockstep on/off) during power-on.

LSM is enabled or disabled by a configuration bit in the DCF client 'Miscellaneous' in UTest flash memory. The checker core shall always be configured to be enabled. [SM_159]If the LSM is disabled, the checker core and the RCCUs are disabled. The checker core will not work independently from the master core. [end][SM_160] No dynamic switching is possible between LSM on and LSM off, and a reset is required to reestablish LSM. [end] Disabling of LSM triggers a fault indication to the FCCU.

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]

Chapter 4
Hardware requirements

4.1 Hardware requirements on system level

This section lists necessary, or recommended, measures on the system level for the MPC5746R to achieve the functional safety goal(s).

The MPC5746R offers an integrated functional safety architecture using dual-core delayed lockstep CPU, replicated function blocks, self-test units and other elements to detect faults. By these means, SPFs and latent failures can 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 MPC5746R, 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 MPC5746R VDD_HV_PMC pin is supplied with a 5 V source as specified in the *MPC5746R Data Sheet*. For most applications, the 1.25 V digital core supply is generated by an external ballast transistor from either a 5 V or 3.3 V source, but can also be supplied by an independent external source. The 5 V supplies generated from the external IC need overvoltage monitoring. Otherwise, the will not be guaranteed to function correctly if the hard voltage limits of the technology are exceeded (see the *MPC5746R Data Sheet* for limits).

The external circuit will also monitor the ERROR[*n*] signals. Through a digital interface (for example, SPI), the MPC5746R 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 MPC5746R. A fail-safe output is also available to control or deactivate any fail-safe circuitry (for example, power switch).

Although ICs providing the described functionality are available from Freescale, 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).

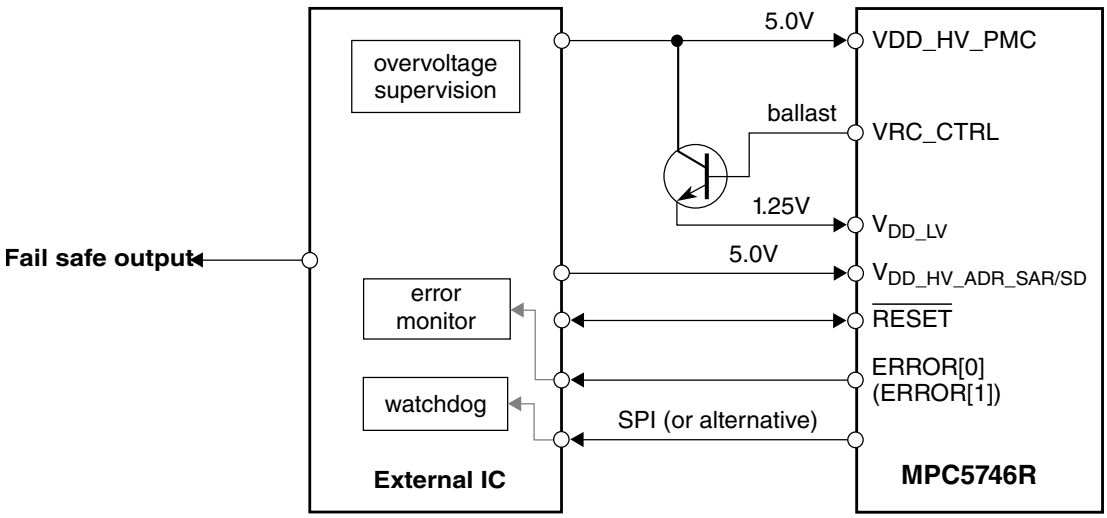


Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the MPC5746R for safety-related applications.

Failure rates of external services are only included for specific circuits (clock, 1.25 V supply) in the FMEDA of the MPC5746R, and must be included in the system FMEDA by the system integrator and user.

4.1.1.1 High impedance outputs

If the MPC5746R is considered to be in a Safe $state_{MCU}$ (for example, unpowered and outputs tristated), the system containing the MPC5746R may not be compliant with the Safe $state_{system}$. A possible system level countermeasure to achieve Safe $state_{system}$ may be to place pull-up or pull-down resistors on I/O when the high-impedance state is not considered safe.

Assumption:[SM_038]If a high-impedance state on an output pin is not safe, pull-up or pull-down resistors shall be added to safety-critical outputs. The need for this will be application dependent for the unpowered or reset (tristated I/O) MPC5746R.[end]

Rationale: In order to bring the functional safety-critical outputs to a level such that a Safe $state_{system}$ is achieved.

4.1.1.2 External Watchdog (EXWD)

Assumption:[SM_039] 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 MPC5746R for safety-related systems. [end]

The trigger may be a discrete signal(s) or message object(s).

Assumption:[SM_040] If within a defined timeout period the EXWD is not triggered, a failure will be considered to have occurred which would then switch the system to a Safe $state_{system}$ within the FTTI. [end]

(For example, the EXWD disconnects the MPC5746R from the power supply, or communication messages are invalidated by disabling the physical layer driver).

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

The implementation of the communication between the MPC5746R 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 MPC5746R, even if it is held in reset. Therefore, in case a voltage range is violated, it is required to either disable power to the MPC5746R or to replace the MPC5746R after an over voltage event (continuously disable safety function).

Assumption:[SM_042] Measures maintaining system level Safe $state_{system}$ during and after any supply voltage above the specified operational range is required. The *MPC5746R Microcontroller Data Sheet* provides specific operating voltage ranges that must be maintained. [end]

Assumption:[SM_086] It is assumed that external power of the appropriate voltage is supplied. [end]

Assumption:[SM_087] It is assumed that the external power is supervised for high and low deviations. [end]

Assumption:[SM_088] It is assumed that the MCU is kept in reset if the external voltage is outside specification and is protected against voltage over the maximum survivable voltage of the technology. [end]

Recommendation: On the system level, in order to avoid a situation where an over-voltage will be supplied to the MPC5746R, permanently disable (Safe_state$_{system}$) the system when an over-voltage is recognized.

Implementation hint: An external and independent device may provide an over voltage monitor for the external 5.0 V MPC5746R supplies. If the supplied voltage level is above the recommended operating voltage range of the MPC5746R, the MPC5746R should be maintained with no power. The external power supply monitor will switch the system to a Safe state$_{system}$ within the FTTI, and maintain it in Safe state$_{system}$ (for example, over-voltage protection with functional safety shut-off, or a switch-over to a second power supply unit).

If the MPC5746R 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 the 1.25 V core supply can be detected by the MPC5746R itself if the core supply LVD has been enabled by properly setting the related DCF record, but system level measures might be required to maintain the Safe state$_{system}$ in case an over-voltage situation may cause damage to the MPC5746R.

Some internal voltage monitors can be enabled or disabled using DCF records. Refer to the MPC5746R datasheet to identify which ones can be disabled and their configuration of the internal voltage monitor elements.

4.1.1.4 Error Out Monitor (ERRM)

If the MPC5746R signals an internal failure on the FCCU error out signals (ERROR[0], and optionally ERROR[1]), the system may no longer rely on the integrity of the other MPC5746R outputs for safety functions. If an error is indicated, the system has to switch to, and remain in, Safe state$_{system}$ without relying on the MPC5746R. Depending on its functionality, the system might disable or reset the device as a reaction to the error indication (see **Assumptions** in Safe state).

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

- Both FCCU error out signals connected to the external device
- Only a single FCCU error signal connected to the external device

Assumption:[SM_043]The overall system needs to include measures to monitor ERROR[*n*] of the MPC5746R and move the system to a Safe $state_{system}$ when an error is indicated.[end]

4.1.1.4.1 Both FCCU signals connected to separate device

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

This configuration does not require any dedicated software support.

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

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]

Implementation hint: Monitoring the ERROR[0] and ERROR[1] through asynchronous combinatorial logic (for example, XOR gate) can generate glitches. Synchronous sampling, or asynchronous oversampling, these signals reduces the possibility of glitches.

4.1.1.4.2 Single FCCU signal connected to separate device

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

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

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

- ERROR[0] (or ERROR[1]) read back internally.
- ERROR[0] (or ERROR[1]) connected externally to a GPIO.

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

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

The advantage of a single 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 ERROR[0], or ERROR[1], is connected to a device not using time domain coding, the ERROR[*n*] signal needs to be verified that it is working properly before starting execution of any safety function.

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

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

Since the FCCU will be monitoring the system, it is sufficient to check ERROR[0] (or ERROR[1]) 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 MPC5746R in a single error output configuration, the application software will need to configure the signals, and pads, adjacent to ERROR[0] (or ERROR[1]) to have a lower drive strength, and the error output signal is configured with highest drive strength. [end]

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

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

Rationale: Decode the time domain coding

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

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

4.1.2 Optional hardware measures on system level

As Input/Output operations are highly application dependant, functional safety assessments of them are not effective on a SEooC level. Functional safety of Input/Output modules and peripherals may be assessed on a system level. The following sections provide examples of possible functional safety mechanisms regarding some Input/Output operations.

Assumption under certain conditions:[SM_044] When data communication is used in the implementation of a safety function, 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.

4.1.3 PowerSBC

The system basis chips MC33907 and MC33908 (PowerSBC) from Freescale are ideally suited to be used in combination with MPC5746R 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 MPC5746R.

Out of a single battery supply with a wide voltage range (V_{SUP}, 3.5 V…28 V), the MC33907/08 generates 5 V (V_{CCA}) and 3.3 V (V_{CORE}) to supply the MPC5746R as well as an auxiliary voltage (V_{AUX}) to supply other devices (for example, sensors or separate ICs). The 1.2 V for digital core supply is generated by an external ballast transistor from V_{CORE}. All voltages generated in the MC33907/08 are independently monitored for under and over voltage.

The MC33907/08 also monitors the state of the error out pins and , using the bistable protocol. Via SPI, the MPC5746R 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 MPC5746R. 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 MPC5746R.

By a connection of the signal MUX_OUT to an ADC input of MPC5746R, 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 MPC5746R directly with a CAN or LIN bus.

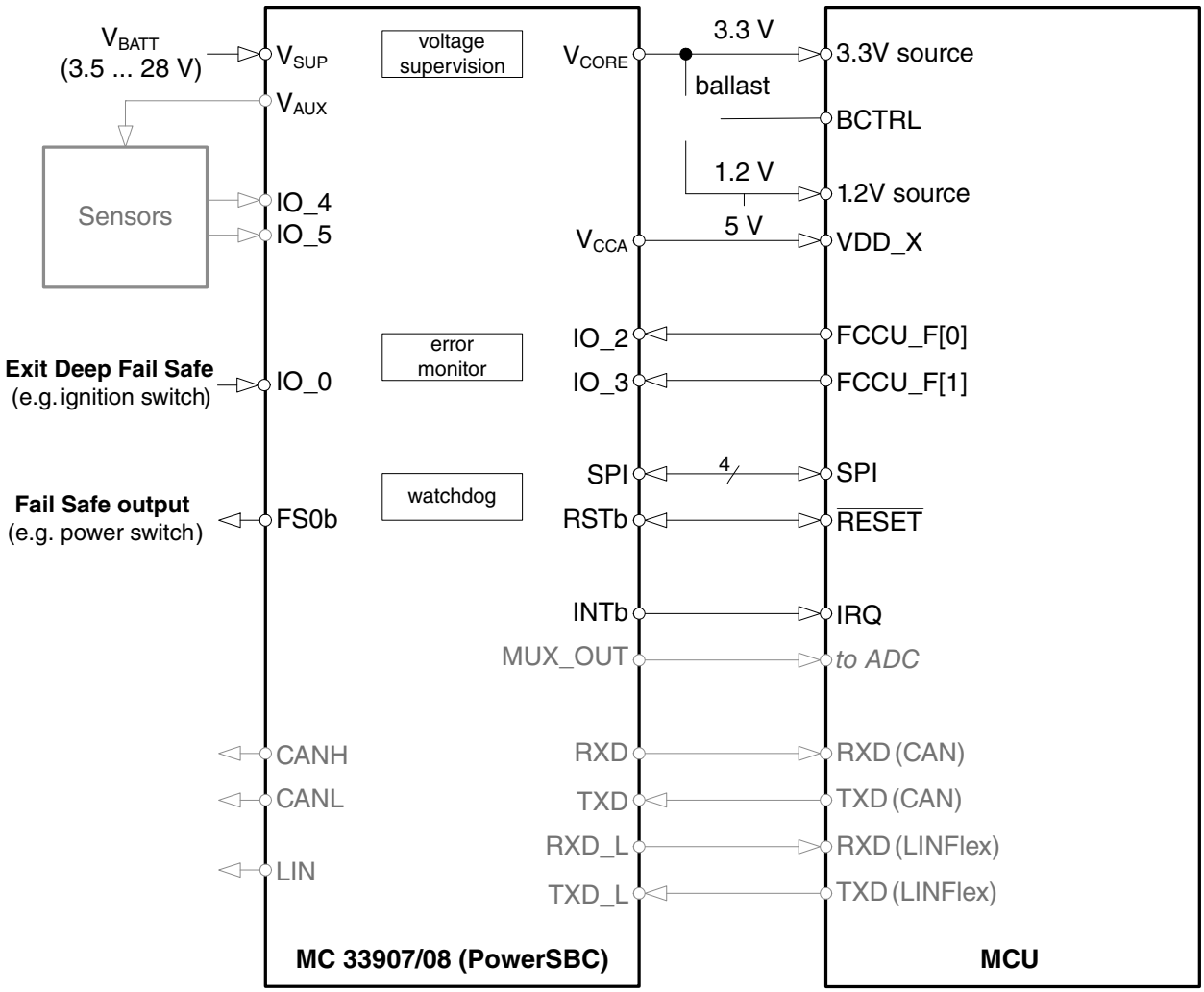


Figure 4-2. Functional safety application with PowerSBC

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 MPC5746R.

Given the application independent nature of the MPC5746R, no general safety function can be specified. To define a specific safety function the MPC5746R 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 used 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 must ensure that the functions described in this section are not activated while running functional safety-relevant operations.

5.1.1.1 Debug mode

The debugging facilities of the MPC5746R 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 MPC5746R may not enter debug mode. The state of the JCOMP signal determines whether the system is being debugged or whether the system operates in normal operating mode. When JCOMP is logic low, the JTAGC TAP controller is kept in reset for normal operating mode. When it is logic high, the JTAGC TAP controller is enabled to enter debug mode. During boot, measures must be taken to ensure that JCOMP is not be asserted by external sources so entering debug mode can be avoided. [SM_046]The activation of debug mode, if JCOMP is low (for example, due to hardware failures), is supervised by the FCCU, and it will signal a fault condition when debug mode is entered. [end] If the FCCU recognizes erroneous activation of debug mode, JTAG signals will no longer recognize any input as being legal debug commands.

Assumption:[SM_047]Debugging will be disabled in the field while the MPC5746R is being used for safety-relevant functions.[end]

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

Rationale: To improve resilience against erroneous activation of debug mode

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

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

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 MPC5746R in the debug mode.

FlexCAN_ MCR[FRZ] controls FlexCAN Module behavior in the debug mode. If the FlexCAN_ MCR[FRZ] = 0, the FlexCAN Module continues communication (not affected by debug mode) when the MPC5746R in the debug mode.

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

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

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

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

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

5.1.1.2 Test mode

Several mechanisms of the MPC5746R can be circumvented during test mode which endangers the functional safety integrity.

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

Recommendation: Use system level software measure to disable test mode.

Implementation hint: The TESTMODE pin is for test purposes only, and must be tied to GND during normal operating mode. From a system level point of view, measures must ensure that the TESTMODE pin is not connected to V_{DD} during boot to avoid entering test mode. [SM_050] The activation of test mode is supervised by the FCCU and will signal a fault condition when test mode is entered. [end]

5.2 MPC5746R modules

5.2.1 Fault Collection and Control Unit (FCCU)

The FCCU uses a hardware fail safe interface which collects faults and brings the MPC5746R 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 MPC5746R into an appropriately configured Safe $state_{MCU}$. To achieve this,

application software only has to configure the FCCU appropriately. No CPU intervention is required for collection and control operation, unless the FCCU is specifically configured to cause software intervention (by triggering IRQs or NMIs).

[SM_052]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: [end]

- Collection of redundant hardware checker results (for example, the RCCU, see Redundancy Control Checking Unit)
- Collection of error information from modules whose behavior is essential to the functional safety goal
- Configurable and graded fault control:
  - Internal reactions
    - No reset reaction
    - IRQ
    - Functional Reset
  - External reaction (external failure reporting using ERROR[*n*])

The FCCU is checked by the FCCU Output Supervision Unit (FOSU) which provides a secondary path for failure indication and reports to the Reset Generation Module (MC_RGM). The FOSU only causes a reset when the FCCU does not react to the incoming failure indication. The FOSU cannot be configured in any way, but it defines a maximum time (1000h IRCOSC cycles) that the FCCU can be held in the configuration state.

Please see the table "FCCU Fault Inputs" in the Reference Manual for sources for critical faults to be signaled to the FCCU and the type of issued reset.

The FCCU has two external signals, ERROR[0] and ERROR[1], through which critical failures are reported. When the MPC5746R is in reset or unpowered, these outputs are tristated.

Assumption:[SM_294] If the MPC5746R is used in a single error out pin mode, ERROR[0] will be the error out pin. [end]

[SM_292] If the application software is using the MPC5746R in a single error out pin mode, the software will configure pins neighboring ERROR[0] to use a lower drive strength, and will configure ERROR[0] to use the highest drive strength. [end]

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 MPC5746R, or an actuator, from the power supply).

5.2.1.1 Initial checks and configurations

Aside from the possible initial configuration, no intervention from the MPC5746R is necessary for fault collection and reaction.

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

Rationale: Maintain the MPC5746R in the Safe $state_{system}$ in case of failure

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

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

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

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

Assumption:[SM_166] If the MPC5746R signals an internal failure via its error out signals (ERROR[*n*]), the system can no longer safely use the MPC5746R safety function outputs. If an error is indicated, the system has to be able to remain in Safe $state_{system}$ without any additional action from the MPC5746R. Depending on its functionality, the system might disable or reset the MPC5746R as a reaction to the indicated error. [end]

5.2.1.2 Runtime checks

Assumption under certain conditions:[SM_055] If the MPC5746R is continuously switching between a standard operating state and reset, or fault state, without a MPC5746R shutdown, system level measures must be implemented to ensure that the system meets the Safe $state_{system}$ criteria. [end]

Implementation hint: Software may be implemented to reduce the likelihood of cycling between a functional and fault states. For example, in the case of periodic non-critical faults, the software could clean the respective status and periodically move the MPC5746R from a fault state to normal state. This procedure may help avoid the possible looping between functional and fault states.

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

The application software should 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_148] Before resetting the reset counters, the application software shall ensure that it can detect longer reset cycles caused by faults in normal operation. [end]

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

5.2.2 Reset Generation Module (MC_RGM)

5.2.2.1 Initial checks and configurations

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

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

To trigger a reset of the MPC5746R by software, the MC_ME_MCTL[TARGET_MODE] shall be used. Writing MC_ME_MCTL[TARGET_MODE] = 0000b causes a functional reset where writing MC_ME_MCTL[TARGET_MODE] = 1111b causes destructive reset (see section "Reset Generation Module (MC_RGM)" of the *MPC5746R Reference Manual* for details).

Recommendation: The peripheral access control in the PBRIDGE*n* should be configured by application software to prohibit access to the MC_RGM_PRST[*n*] (individual module reset programming model).

5.2.2.1.1 Consecutive resets

[SM_057]Permanent cycling through otherwise safe states or permanent cycling between a safe state and an unsafe state is considered a violation of the safety goal. Specifically, this scenario relates to a continuous Reset-Start, Operation-Reset or Reset-Selftest-Reset sequence. Allowing such cycles would be problematic as it would allow an unlimited number of attempts. [end]

[SM_058]To detect a loop of resets, the MPC5746R supports functional reset escalation which can be used to generate a destructive reset if the number of functional resets reaches the programmed value. [end] Once the functional reset escalation is enabled, the Reset Generation Module (MC_RGM) increments a counter for each functional reset that occurs between writes to the MC_RGM_FRET register. When the number of functional resets reaches the programmed value in the MC_RGM_FRET, the MC_RGM initiates a destructive reset. The counter can be cleared by software, destructive reset or power-on reset.

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

Assumption:[SM_060] Since the default setting for the reset counter is disabled, the SW must enable the counter by writing a non-zero value to MC_RGM_FRET register. [end]

5.2.3 Self Test Control Unit (STCU2)

The STCU2 executes built-in self-test (LBIST, MBIST) and gives reaction to detected faults by signaling faults to the FCCU (see "Self-Test Control Unit (STCU2)" in the *MPC5746R Reference Manual* for details).

5.2.3.1 Initial checks and configurations

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

Assumption under certain conditions: [SM_062] When built-in self-test (for example, LBIST, MBIST, ABIST) circuits of the MPC5746R are used as functional safety integrity measures (for example, to detect random faults, latent fault detection, and single-point fault detection) in a functional safety system, functional safety integrity measures on a system level shall be implemented, ensuring STCU2 integrity during/after STCU2 initialization, but before executing a safety function. [end]

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

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

Implementation hint: System (application) level software shall carry out checking of STCU2 for ensuring STCU2 integrity (see sections "Off-Line Self-Test Sequence" and "On-Line Self-Test Sequence" in "Self-Test Control Unit (STCU2)" chapter in the *MPC5746R Reference Manual*).

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

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

This is an additional functional safety layer since the STCU2 propagates the LBIST/ MBIST and internal faults to the MC_RGM or the FCCU. So, reading STCU2_LBS, STCU2_LBE, STCU2_MBSL, STCU2_MBSH, STCU2_MBEL, STCU2_MBEH and STCU2_ERR registers helps increase the STCU2 self-test coverage.

Implementation hint: The STCU2 shall be configured (in UTest 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 *MPC5746R Reference Manual*).

5.2.4 Temperature Sensors (TSENS)

The MPC5746R has two temperature sensors that are read from the ADC modules and the redundant temperature sensors are in separate safety lakes. Each temperature sensor generates one analog voltage which is proportional to the absolute current junction temperature of the MPC5746R and three digital outputs that signal whether the junction temperature has reached either a preset low temperature threshold or one of two preset high temperature thresholds.

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

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

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

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

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

Implementation hint: When the temperature threshold detection feature is enabled, the temperature sensor monitors the internal junction temperature of the MPC5746R and asserts a signal if any of the following three specified temperature thresholds are crossed:

- Low temperature digital output – signals if the junction temperature falls below the low temperature threshold (-40 °C)
- High temperature digital output 1 – signals if the junction temperature rises above the high temperature threshold (150 °C)

[SM_063]Two temperature sensors monitor the substrate temperature to detect over-temperature conditions before they cause any CMFs (for example, faults due to over-temperature which causes identical erroneous results from both cores). [end] The maximum operating junction temperature is specified in the *MPC5746R Data Sheet*. The sensor output is forwarded to the appropriate ADC 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_064] Application software shall configure the FCCU and the PMC registers related to temperature sensor configuration to react to over-temperature faults of the temperature sensors (see the "FCCU Fault Inputs" table in the 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 TSENS*n*. [end]

Assumption:[SM_066] During power up, the two temperature sensors are to be read by software. The software must 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

The objective of the Software Watchdog Timer (SWT) is to detect a defective program sequence when individual elements of a program are processed in the wrong sequence, or in an excessive period of time. Once the SWT is enabled, it requires periodic and timely execution of the watchdog servicing procedure. The service procedure must be performed within the configured time window, before the service timeout expires. [SM_161] 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. [end] [SM_162] 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 in the Reference Manual). [end]

Assumption:[SM_067] These requirements apply to the SWT for safety-related applications: [end]

- The SWT must to be enabled and configuration registers have to be hard-locked against modification.
- The SWT time window settings must be set to a value less than the FTTI.
    - Detection latency shall be smaller than the FTTI.
- Before the safety function is executed, software must verify that the SWT is enabled by reading the SWT control register (SWT_CR).

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) must contain a 32-bit value that represents a timeout less than the FTTI.

Assumption: [SM_068] In general, it is expected that the SWT helps to detect lost or significantly slow clocks. Thus, the SWT needs to be used to also detect hardware faults, not only to detect software faults. [end] 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 MPC5746R provides the hardware support (SWT) to implement both control flow and temporal monitoring methods. If Windowed mode and Keyed Service mode (two pseudorandom key values used to service the watchdog) are enabled, it is possible to reach a high effective temporal flow monitoring.

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

Two service procedures are available:

- [SM_163] A fix service sequence represented by a write of two fix values (A602h, B480h) to the SWT service register. Writing the service sequence reloads the internal down counter with the timeout period. [end]
- [SM_164] 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 $SK_{(n+1)} = (17 \times SK_{(n+3)}) \bmod 2^{16}$. [end]

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

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

The task of the Redundancy Control Checking Unit (RCCU) unit is to perform a cycle-by-cycle comparison of the outputs between the master and checker cores and the master and checker eDMA units, respectively. The error information is forwarded to the FCCU. The RCCUs are automatically enabled when MPC5746R is in LSM mode.

NOTE

On the MPC5746R, 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 MPC5746R. The RCCU cannot be disabled by application software. Consequently, the respective FCCU input should not be disabled.

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

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 MPC5746R.

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

5.2.7.1 Runtime checks

Parts of the MPC5746R configuration registers do not provide the functional safety integrity ISO 26262 requires for high functional safety integrity targets on their own. This relates to systematic faults (for example, application software incorrectly overwriting registers), as well as random hardware faults (bit flipping in registers).

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

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

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

Assumption:[SM_071] Safety software running on the safety core must check correct initialization of the MPC5746R 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: The CRC module offloads the CPU in computing a CRC checksum. The CRC has the capability to process two different CRC calculations at the same time. To verify the content of the MPC5746R 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 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:

- SIUL_SWTEST_REGCRC

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

- ADC0_SWTEST_REGCRC

  The ADC0 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- ADC1_SWTEST_REGCRC

  The ADC1 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- ADC2_SWTEST_REGCRC

The ADC2 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- ADC3_SWTEST_REGCRC

  The ADC3 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

- CTU0_SWTEST_REGCRC

  The CTU0 configuration registers are read and a CRC checksum is computed. The checksum is compared to the expected value.

5.2.8 IRCOSC

The 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 (see the *MPC5746R Data Sheet*). 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_073] The IRCOSC frequency is measured and compared to the expected frequency of 16 MHz. This test is performed after power-on, but before executing any safety function. Software writes CMU_CSR[SFM] = 1 to start the frequency measurement, and the status of the measurement is checked by reading this same field. When as CMU_CSR[SFM] = 0 the frequency measurement has completed (see "Frequency meter" section in the "Clock Monitor Unit (CMU)" chapter of the *MPC5746R 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_074] To detect failure of the IRCOSC, the application software shall utilize the CMU's frequency meter to read the IRCOSC frequency and compare it against the expected value of 16 MHz[1].[end]

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

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

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

5.2.9 External Oscillator (XOSC)

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

5.2.9.1 Initial checks and configurations

Assumption:[SM_075] FlexCAN should not be clocked directly by the XOSC in normal operation unless the effects of clock glitches are sufficiently detected by the applied FT-COM layer.[end]

1. Nominal frequency of the IRCOSC is 16 MHz, but the post trim accuracy over voltage and temperature must be taken into account (see the *MPC5746R Data Sheet*).

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 into the operational state.[end]

5.2.10 Dual PLL Digital Interface (PLLDIG)

The MPC5746R consists of two PLLs used to generate high speed clocks, an FMPLL (PLL1) (which provides a frequency modulated clock) and non-FMPLL (PLL0). [SM_077] The FMPLL and non-FMPLL provide a loss of lock error indication that is routed to the MC_RGM and the FCCU. [end] 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 MPC5746R modules. The auxiliary clock from the non-FMPLL is instead distributed to those peripherals that require precise timing, its clock is not modulated.

5.2.10.1 Initial checks and configurations

After system reset, the external crystal oscillator is powered down and the PLLs are deactivated. Software shall enable the oscillator.The MPC5746R uses after system reset the internal RC oscillator clock (IRCOSC) as clock source (see the "Oscillators" chapter in the *MPC5746R Reference Manual* and IRCOSC for details on IRCOSC configuration).

Assumption:[SM_078] Before executing any safety function, a high quality clock (low noise, low likelihood for glitches) based on an external clock source shall be configured as the system clock of the MPC5746R. [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. In general MC_CGM_AC3_SC[SELCTL] and MC_CGM_AC4_SC[SELCTL] shall be set to 1.

Assumption under certain conditions:[SM_079] When clock glitches endanger the system level functional safety integrity measures, 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 stemming from the external crystal and its hardware connection to the MPC5746R.

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

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

5.2.11 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 MPC5746R is still running on the IRCOSC.

The CMUs are driven by the 16 MHz internal reference clock oscillator (IRCOSC) to ensure independence from the monitored clocks. CMUs flag errors associated with conditions due to clock out of a programmable bounds and loss of reference clock. If a supervised clock leaves the specified range for the MPC5746R, an error signal is sent to the FCCU. [SM_298] The CMU must be configured by the application software to send notice of a clock failure to the FCCU, such that the FCCU can force a reset. [end]

The MPC5746R includes the CMUs shown in Table 5-1.

Table 5-1. Clock Monitoring Units

CMU	Monitored Clock
CMU_FXBAR	Fast Crossbar
CMU_SXBAR	Slow Crossbar
CMU_AIPS	Peripheral Bus
CMU_PER	Peripheral Clock
CMU_ADCSD	Sigma Delta ADC
CMU_SARADC	SAR ADC
CMU_SENT	SENT Module
CMU_EMIOS	eMIOS Module
CMU_ETPU2	ETPU2 Module
CMU_CLKOUT	CLKOUT Signal
CMU_PLL	PLL module

[SM_156]All CMUs use the IRCCOSC (16 MHz internal oscillator) as the reference clock for independent operation from the monitored clocks. Their purpose is to check for error conditions due to: [end]

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

The CMUs supervise the frequency range of various clock sources. [SM_157]In case of abnormal behavior, the information is forwarded to the FCCU as faults (please see the table "FCCU Fault Inputs" in the Reference Manual for details). [end]

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

5.2.11.1 Initial checks and configurations

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

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

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_PLL (CMU_0). Software configures CMU_PLL_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[HFREF]) and low frequency reference (CMU_*n*_LFREFR[LFREF]) is configured depending on CMU_PLL (CMU_0).

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

5.2.12 Mode Entry (MC_ME)

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

Implementation hint: The SWT may provide the time monitoring.

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

5.2.13 Power Management Controller (PMC)

The PMC manages the supply voltages for all modules on the MPC5746R. It includes the internal regulator for the logic power supply (1.25 V) and a set of voltage monitors (low voltage detectors (LVD) and high voltage detectors (HVD)). If one of the monitored voltages goes below (LVD) or above (HVD) a given threshold, a destructive reset is initiated to control erroneous voltages before they cause a CMF (for correct operating voltage ranges see the *MPC5746R Data Sheet*).

Some LVD and HVD can be configured to send a functional reset or an interrupt. However, the safety relevant LVD/HVD are not configurable, and will always send a destructive reset (see PMC_REE and PMC_RES register descriptions in the "Power Management Controller digital interface (PMC_dig)" chapter of the *MPC5746R Reference Manual*).

To ensure functional safety, the PMC monitors various supply voltages of the MPC5746R (as seen in Table 5-2).

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

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

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

Apart from the self-test, the use of the PMC for safety-related applications is transparent to the user because the operation of the PMU is automatic.

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

Undervoltage and overvoltage conditions are primarily reported to the MC_RGM, where they directly cause a transition into a safe state by a reset. This solution was chosen because safety-relevant voltages have the potential to disable the failure indication mechanisms of the MPC5746R (the FCCU). The LVDs and HVDs also report errors to the FCCU, but since the LVD and HVD errors are handled by the MC_RGM, the FCCU error reporting is not utilized.

Note

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

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

If the power supply is out of range, MPC5746R shall be kept under 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 MPC5746R even if kept in reset.

Table 5-2. PMC monitored supplies

Detector Type	Detector Name	Voltage Monitored
VDD_HV_FLA	LVD_FLASH, HVD_FLASH	3.3 V Flash supply
VDD_HV_IO_MAIN	LVD_IO	5.0 V I/O supply
VDD_HV_PMC	LVD_PMC, HVD_HV	5.0 V PMC supply
VDD_LV_CORE	LVD_core_hot, LVD_core_cold, HVD_core	1.25 V core supply
VDD_HV_ADV_SAR	LVD_SAR	5.0 V SAR ADC supply

Additionally, the SD ADC supply and reference and IO segment supplies can be monitored using the SAR ADC. See the MPC5746R Reference Manual for further details.

5.2.13.1 1.25 V supply supervision

Voltage detectors LVD_core_cold, LVD_core_hot, and HVD_core monitor the digital (1.25 V) core supply voltage for over and under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case the core main voltage detector detects over or under voltage during normal operation of the MPC5746R, a destructive reset is triggered.

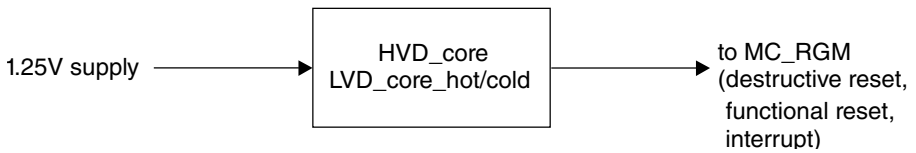


Figure 5-1. Logic scheme of the core voltage detectors

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

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

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

Implementation hint: To reduce the likelihood of destructive damage due to a stuck-closed external ballast transistor (item/system level component), it may be necessary to implement two ballast transistors sequential as a system level functional safety integrity measure. This will load the regulator with two ballast transistors. In order to use two ballast transistor a ~ 30 % Cg (or smaller, transistor gate capacity) should be selected. Alternative the digital (1.25 V) core supply voltage may be monitored externally and the power supply shut-down in case of an over voltage. Alternatively an external 1.25 V HVD may detect over voltage and shut down the supply voltage.

5.2.13.2 3.3 V supply supervision

Voltage detector LVD_VFLASH monitors the 3.3 V supply from the internal 3.3 V regulator for under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detectors. In case a single LVD detects under voltage during normal operation of the MPC5746R, a destructive reset is triggered.

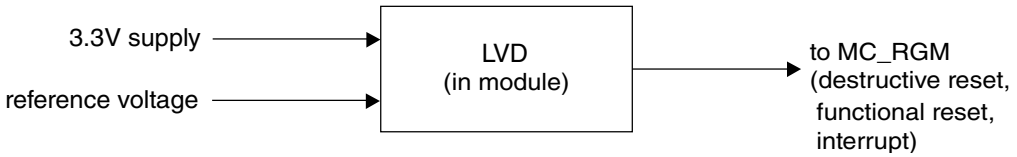


Figure 5-2. Logic scheme of the 3.3 V voltage detectors

5.2.14 Memory Protection Units

As a multi-master, concurrent bus system, the MPC5746R provides safety mechanisms to prevent non-safety masters from interfering with the operation of the safety core. MPC5746R also contains mechanisms to handle the concurrent operation of software tasks with different or lower ASIL classifications.

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

5.2.14.1 Core Memory Protection Unit (CMPU)

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

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

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

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

5.2.14.2 System Memory Protection Unit (SMPU)

The System MPU (SMPU) provides memory protection at the crossbar (XBAR). The SMPU splits the physical memory into 16 different regions. Each XBAR master (Core, DMA, SIPI) can be assigned different access rights to each region. [SM_093]The SMPU can be used to prevent non-safety masters (including DMA) from accessing restricted memory regions. [end]

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

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

5.2.14.3 Initial checks and configurations

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

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

Implementation hint: The MPUs shall be used to ensure that only authorized software routines can configure modules and all other bus masters can access only their allocated resources according to their access rights.

5.2.15 PBRIDGE protection

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

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

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

5.2.15.1 Initial checks and configurations

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

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

5.2.16 Built-in Hardware Self-Tests (BIST)

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

[SM_096] To ensure absence of latent faults, the self-test executes both Logic Built-In Self Test (LBIST) and Memory Built-In Self Test (MBIST) during boot while the MPC5746R is still under reset (offline). [end] The boot time BIST includes the scan-based LBIST to test the digital logic and the MBIST to test all RAMs and ROMs.[2]

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

2. This does not include flash memory.

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

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

Note

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

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

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

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

The MPC5746R 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)
- ADC (PBIST, executed at runtime or executed at least once per FTTI), latent failure measure and single-point 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)

During boot, tests (MBIST, LBIST) are performed after the occurrence of a destructive or external reset (unless they are disabled). All tests during boot are executed before application software starts executing. If the tests fail, the MPC5746R will remain in Safe $state_{MCU}$.

All tests may be performed without dedicated external test hardware.

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

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

5.2.16.1 MBIST

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

NOTE

In principle MBIST can be run at any time, but the MPC5746R will execute a reset after MBIST completes.

5.2.16.2 LBIST

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

NOTE

In principle LBIST can be run at any time, but the MPC5746R will execute a reset after LBIST completes.

5.2.16.3 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).

5.2.16.4 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.

5.2.16.5 Peripheral Built-In Self-Test (PBIST)

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

5.2.17 End-to-end ECC (e2eECC)

The MPC5746R includes end-to-end ECC (e2eECC) support for improved functional and transient fault detection capabilities. Memory protected by 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 MPC5746R e2eECC protected memory, the bus master initiates the data write and generates ECC checkbits based on 29-bit address and 64-bit data field for the computational shell (32-bit data field for the peripheral shell). 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 a 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 array, RAM array, CAN RAM, DMA RAM array.
- ECC on address and data
  - Single Error Correction/Double Error Detection (SECDED) covers 64-bit data and 29-bit address bits for the computational shell (32-bit data field for the peripheral shell).

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 account for 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.

5.2.18 Interrupt Controller (INTC)

The Interrupt Controller (INTC) provide 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 ones. No specific hardware protection is provided to reduce the likelihood of spurious or missing interrupt requests, caused by

faults before the IRQ, such as by Electromagnetic Interference (EMI) on the interrupt lines, bit flips in the interrupt registers of the peripherals, or a fault in the peripherals. The Interrupt Controller (INTC) can drop, delay or create spurious interrupts.

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

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.

5.2.18.1 Periodic low latency IRQs

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

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

5.2.18.2 Non-Periodic low latency IRQs

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

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

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

5.2.18.3 Runtime checks

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

Rationale: To manage spurious or missing interrupt requests.

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

5.2.19 Enhanced Direct Memory Access (eDMA)

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

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

5.2.19.1 Runtime checks

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

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

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

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

Assumption under certain conditions:[SM_102] Applications that are not resilient to spurious, or missing functional safety-relevant, eDMA requests 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

5.2.19.1.1 Peripheral lake eDMA transfers

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

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

For example, if eDMA Channel Mux 1 is faulty and thus disturbs access to a peripheral in its lake, then an access triggered by eDMA Channel Mux 0 to a peripheral in its own lake will not be faulty and thus show a deviation from the faulty transfer.

5.2.19.1.2 Non-replicated eDMA transfers

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

Assumption: [SM_104] If safety-relevant software is using the eDMA to transfer data to a non-replicated peripheral or within the RAM, the following holds:[end]

- [SM_090]Preferably, "always on" channels of the eDMA Channel Mux shall not be used. Instead, the eDMA shall be triggered by software.[end]
- [SM_091] If "always on" channels are used, their failure has to be detected by software. In this case, software must ensure that the eDMA transfer was triggered as expected at the correct rate and the correct number of times. This test shall detect unexpected, spurious interrupts. [end]

5.2.20 System Timer Module (STM)

5.2.20.1 Runtime checks

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

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_105] At every STM interrupt, the IRQ handler shall compare the elapsed time since the previous interrupt versus a free running counter to check whether the interrupt time is consistent with the STM setting. [end]

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

5.2.21 Periodic Interrupt Timer (PIT)

5.2.21.1 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_108] If the PIT is used by the application software in a safety function, a checksum of its configuration registers using the CRC must be calculated and compared with the expected one to verify that the PIT configuration is correct. [end]

The application software shall invoke this test once per FTTI.

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

5.2.22 System Status and Control Module (SSCM)

5.2.22.1 Initial checks and configurations

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

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

Implementation hint: Execution of BAF code may be inhibited by writing SSCM_ERROR[PAE] = 1. Each access to the BAF memory area then produces an exception. This prevents accidential execution of the (non-safety-related application) BAF code.

NOTE

The BAF will not execute on its own during a 'normal' boot of the MPC5746R, but only if a serial boot, a JLR, or a test pattern load is requested.

5.2.23 Memory Error Management Unit (MEMU)

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

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

5.2.23.1 Initial checks and configurations

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]

The application software shall check the MEMU to determine whether the MBIST error was a single-bit vs. multi-bit, the address of the error and the bit position of the error if the error was single-bit.

5.2.23.2 Runtime checks

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

Assumption:[SM_110] Within the FTTI, application software will detect permanent multi-bit error sources after a new ECC error in RAM is reported. Also, within the FTTI, application software shall detect permanent multi-bit error sources caused by multiple address selections.[end]

The software test fetches the error address of corrected errors from the MEMU and assesses the nature of the fault by writing and then reading back a few patterns to the faulty location (SW is available from Freescale Semiconductor).

5.2.24 Flash memory

MPC5746R provides 4256 KB of 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. [SM_112] Array Integrity self-check calculates a MISR signature over the array content and thus validates the content of the array as well as the decoder logic. [end] The calculated MISR value depends on the array content and must be validated by application software.

Implementation hint: The array integrity self check and the ECC logic check must 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. The programmed patterns with one resp. two wrong bits in the UTEST flash memory do not provide any coverage of the ECC logic itself, but can be used in case any additional coverage of the ECC error reaction is path to the MEMU.

5.2.24.1 EEPROM

MPC5746R provides ten blocks (2 x 64 KB and 8 x 16 KB) of the flash memory for EEPROM emulation. ECC events detected on accesses to the EEPROM flash memory blocks are not reported to the Memory Management Unit (MEMU). Single-bit errors are corrected but not signaled to the MEMU. Multi-bit errors are replaced by a fixed word (representing an illegal instruction) and are also not forwarded to the MEMU.

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

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

5.2.24.2 Initial checks and configurations

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

Implementation hint: 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.

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

Implementation hint: This test may be started by application software: its result may be validated by reading the corresponding registers in the flash memory controller after it has been finished (see "Array integrity self check" section in the "Flash memory" chapter of the *MPC5746R Reference Manual*).

5.2.24.3 Runtime checks

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

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

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

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

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

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

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

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

Implementation hint: The flash memory ECC fault report check is executed in software. The test consists of software reading from the flash memory UTEST area (see "UTEST flash memory map" table in the "Memory map" chapter of the MPC5746R Reference Manual). It is a set of test patterns to test the integrity of the ECC logic fault reporting path to the MEMU and FCCU (executed at start-up, latent failure measure).

5.2.25 Body Cross Triggering Unit (BCTU)

The MPC5746R contains one ADC Body Cross Triggering Unit (BCTU), interfaced between the eTPU2, eMIOS, PIT, and the SAR ADCs.

The BCTU allows automatic generation of SAR ADC conversion requests with minimal CPU intervention. The BCTU generates triggers based on input events from the eMIOS, eTPU2, PIT, and/or external pins).

The trigger can be caused by:

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

The BCTU can be used if the application needs to synchronize the reading of some ADC inputs with MPC5746R events (for example, eMIOS, Timers, eTPU2, and/or external pins).

5.2.25.1 Runtime checks

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

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

For each trigger, an ADC command or command list can be defined.

If the application safety function includes the read of inputs synchronized with events (eTPU2, eMIOS, external signals, or any combination), the system integrator can use the BCTU module for this purpose.

For a detailed description of BCTU operation (triggered and sequential mode), its configuration, and use, see the *MPC5746R Reference Manual*.

5.2.26 Error reporting path tests

It is possible to use fake fault injection to check the correct operation of several reporting paths from supervisors to the MEMU. See the "FCCU Fault Inputs" table in the Reference Manual for more details.

Other measures in that column (except LBIST) can also be used for a full error reporting path check if so desired. It should be noted that LBIST covers the logic of the error reporting path as long as it does not cross an LBIST partition boundary. If that happens, a small amount of logic remains uncovered by the LBISTs.

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

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

A multiple cell failure caused for example, by a neutron or alpha particle or a short circuit between cells may cause three or more bits to be corrupted in an ECC-protected word. As result, either the availability may be reduced or the ECC logic may perform an additional data corruption labeled as single-bit correction. This is prevented within the design of MPC5746R by the use of bit scrambling (column multiplexing) which effects, that physically neighboring columns of the RAM array do not contain bits of the same logical word but the same bit of neighboring logical words. Thus the information is logically spread over several words causing only single-bit faults in each word which can be correctly corrected by the ECC. MPC5746R 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.

5.2.27 Glitch filter

An analog glitch filter is implemented on the reset signal of the MPC5746R. A selectable (WKPU_NCR[NFE0]) analog glitch filter is implemented on the NMI-input. External interrupt sources can be configured to be used with any chip GPIO. Interrupt sources (1

to 32) can be configured to have a digital filter to reject short glitches on the inputs. These filters are used to reduce noise and transient spikes in order to reduce the likelihood of unintended activation of the reset or the interrupt inputs.

NOTE

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

5.2.28 Register Protection module (REG_PROT)

The PowerPC architecture supports two levels of privilege for program execution: user mode and supervisor mode. Only the supervisor mode allows the access to the entire CPU register set, and the execution of a subset of instructions is limited to supervisor mode only. In user-mode, access to most registers including system control registers is denied. It is intended that most parts of the software be executed in user-mode so that the MPC5746R is protected from errant register changes made by other user-mode routines. User versus supervisor mode can also be used as a decision criteria in the MPUs and the peripheral access control (PAC) of the PBRIDGES.

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

- A register cannot be written once soft lock protection is set. The lock can be cleared by software or by a system reset.
- A register cannot be written once hard lock protection is set. The lock can only be cleared by a system reset.
- If neither soft or hard lock is set, the Register Protection module may restrict write accesses for a module under protection to supervisor mode only.

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

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

[SM_285] If registers are protected against random hardware modification, they must also be protected against accidental software writes. [end]

5.2.28.1 Runtime checks

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

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

Implementation hint: Most of the off-platform peripherals have their own Register Protection module. Register Protection address space is inside the memory space reserved for the peripherals (please, refer to the "Register protection (REG_PROT) configuration" section of the MPC5746R Reference Manual). Each peripheral register that can be protected through the Register Protection module has a Set Soft Lock bit reserved in the Register Protection address space. This bit is asserted to enable the protection of the related peripheral registers. Moreover, the Hard Lock Bit (REG_PROT_GCR[HLB] = 1) should be set for best write protection.

5.2.29 Wake-Up Unit (WKPU) / External NMI

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

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

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

5.2.30 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 software execution times, so software countermeasures must detect these faults.

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

5.2.30.1 Runtime checks

[SM_111] The application software shall check the XBAR configuration once after programming[end], but it must also detect failures of the XBAR when safety-relevant functions are running.

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

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

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

5.2.31 Analog to Digital Converter (ADC)

Parts of the Analog-to-Digital Converter (ADC) of the MPC5746R do not provide the functional safety integrity on their own that ISO 26262 requires for high functional safety integrity targets. Therefore system level measures are required.

5.2.31.1 Initial checks and configurations

Assumption under certain conditions:[SM_130] When the Analog-to-Digital Converter (ADC) of the MPC5746R is used in a safety function, suitable system level functional safety integrity measures must be implemented after reset (external reset or destructive reset) before starting the respective safety function to ensure ADC integrity. [end]

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

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

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

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

- CPU mode
- BCTU mode

Assumption: Calibration needs to be completed after destructive reset.

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

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

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

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

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

Implementation hint: Table 5-3 shows mapping of the values stored in "Test flash memory" to be copied into the watchdog registers. Depending on the reference voltage used for the ADCs, ADC*n*_CAL W4 or ADC*n*_CAL W5 is to be used for ADC_*n*_STAW0R.

Please refer to the "self-test analog watchdog" section of the "ADC" chapter and the "Test sector" section of the "Flash Memory" chapter in the *MPC5746R Reference Manual* for details.

Table 5-3. Sample Values for ADC Self-Test Thresholds

ADC register	Step	THRH	THRL	THRH	THRL
STAW0R	S0_3.3V	75Ah	4DFh	1882d	1247d
STAW0R	S0_5.0V	4D0h	2DBh	1232d	731d
STAW1AR	S1(INT)	3h	2h	3d	2d
STAW1BR	S1(FRAC)	3D9h	1E3h	985d	483d
STAW2R	S2	---	FF9h	---	4089d
STAW4R	C0	010h	FF0h	16d	-16d
STAW5R	C1-C11	010h	FF0h	16d	-16d

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

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

Implementation hint: Every hardware BIST is activated via a dedicated command sent to the ADC (see the "SAR ADC" chapter of the *MPC5746R Reference Manual* details on implementing these modes).

The SUPPLY SELF-TEST is executed without interleaved conversion (see *MPC5746R Reference Manual* for details).

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

5.3 I/O functions

[SM_165] The integrity of functional safety-relevant periphery will mainly be ensured by application level measures (for example, connecting one sensor to different I/O modules, sensor validation by sensor fusion, and so on). [end]

Functional safety-relevant peripherals are assumed to be used redundantly in some way. Different approaches can be used, for example, by implementing replicated input (for example, connect one sensor to two 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).

Recommendation: The use of opposing data coding (for example, inversion) is recommended for redundant communication over safety relevant peripherals (for example, DSPI or LINFlexD). Users can choose the approach that best fits their needs.

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_133] Comparison of redundant operation of I/O modules is the responsibility of the application software, as no hardware mechanism is provided for this. [end]

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

Implementation hint: Possible measures could be using different replicated peripherals to implement multiple independent and different channels.

5.3.1 Digital inputs

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

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

5.3.2 Digital outputs

Functional safety digital outputs are always assumed to be written either redundantly or with read back. In case of single output with read back, the feedback loop should be as large as possible to cover faults on system level also. The figure below depicts the connection of two (functional safety critical) actuators connected to the MPC5746R. Actuator 1 is connected to an output peripheral, for example 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 channel. This measure is to confirm, that the generated output signal is correct. This read back may be internally of the MPC5746R (internal read back) or externally (external read back). The external read back covers more types of failures (for example, corrupt wire bonds or solder joints) than the internal read back, but still does not guarantee, that the actuator really behaves as desired. This is achieved by including the actuator and sensor into the read back loop. An alternative solution is to redundantly output a signal. For example, actuator 2 consists of two relays in series to switch off a functional safety-relevant supply voltage. The selection of the suited output connection is part of the I/O functional safety concept on system level.

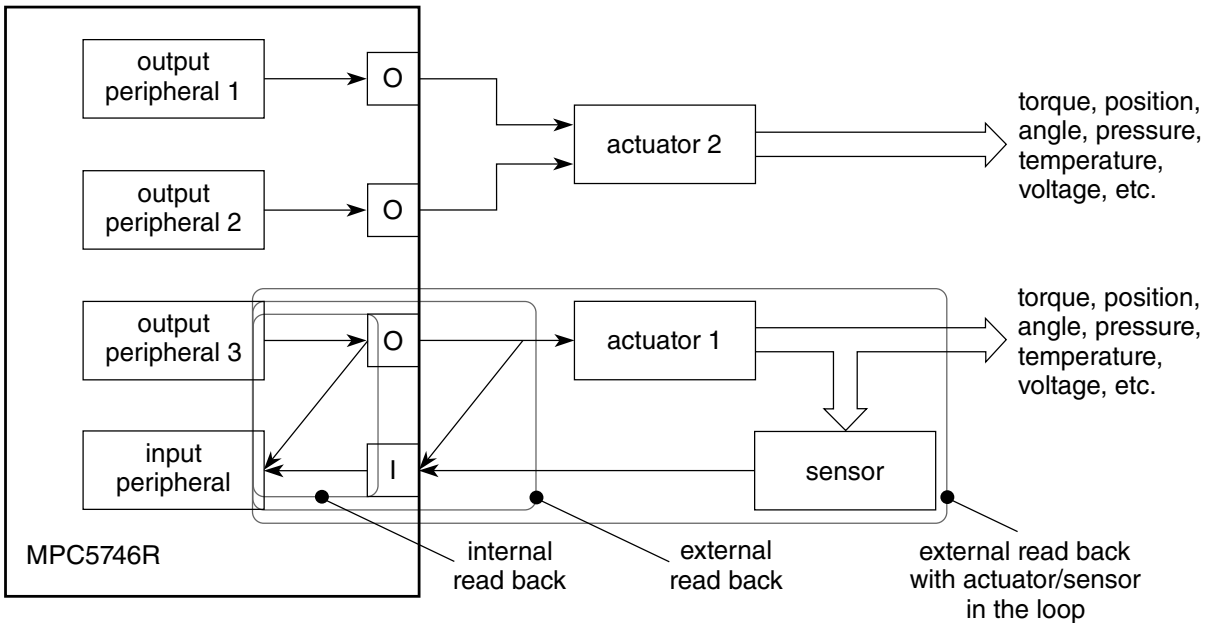


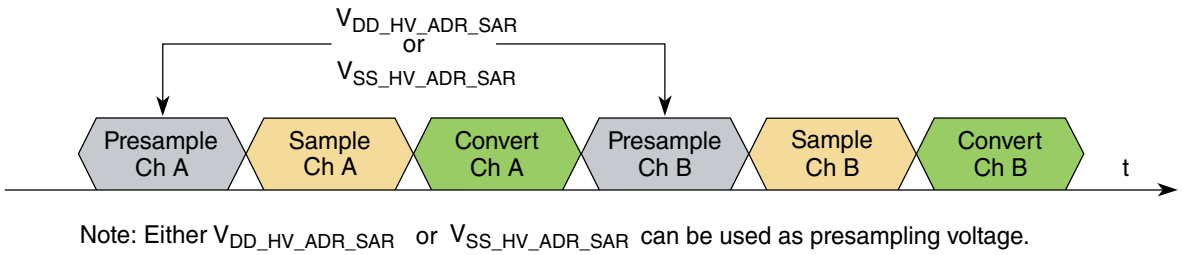
Figure 5-3. Digital Outputs with redundancy and read back

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 deviating from **Safety requirements** as described in the preface.

5.3.3 Analog inputs

5.3.3.1 ADC_SWTEST_TEST1 (open detection)

This test exploits the presampling feature of the ADC. Presampling allows to precharge or discharge of the ADC internal capacitor before it starts the sampling and conversion phases of the analog input received from the pads. During the presampling phase, the ADC samples the internally generated voltage. While in the sampling phase, the ADC samples analog input coming from the pads. In the conversion phase, the last sampled value is converted to a digital value. Figure 5-4 shows the normal sequence of operation for two channels (Presampling – Sampling – Conversion).



Note: Either $V_{DD\_HV\_ADR\_SAR}$ or $V_{SS\_HV\_ADR\_SAR}$ can be used as presampling voltage.

Figure 5-4. Implementation of ADC_SWTEST_TEST1

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

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

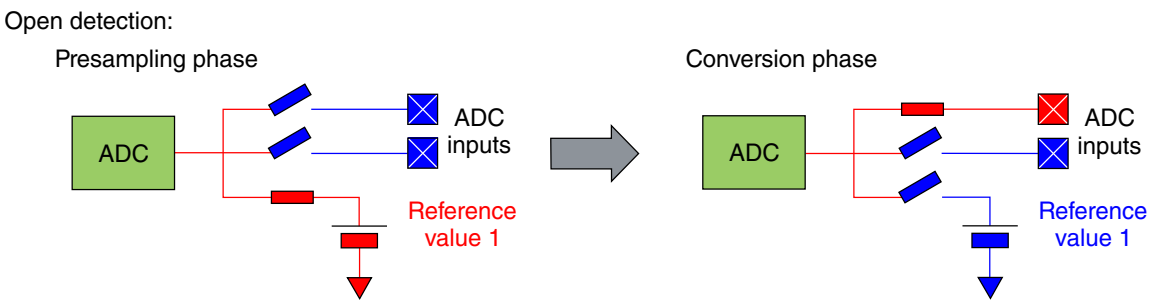


Figure 5-5. ADC_SWTEST_TEST1 (open detection)

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

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

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

Implementation hint: Presampling can be enabled on a per channel basis through the SARADC*n*_PSR0 register. SARADC*n*_PCSR[PREVAL0] selects which reference voltage is used to precharge/discharge the ADC internal capacitor, (SARADC*n*_PSCR[PRECONV] = 0). (See "Analog-to-Digital Converter (SARADC)" chapter in the *MPC5746R Reference Manual* for details on the presampling feature).

Note

Caution! To reduce the likelihood of a false indication of an open fault in the analog multiplexor, signals connected to the ADC inputs should not be outside of the limits of the reference voltages ($V_{DD\_HV\_ADR\_SAR}$, $V_{SS\_HV\_ADR\_SAR}$). In case this

limitation cannot be fulfilled by the application, a more complex algorithm may be necessary (for example, run the test three times with $V_{DD\_HV\_ADR\_SAR}$, $V_{SS\_HV\_ADR\_SAR}$, $V_{DD\_HV\_ADR\_SAR}$).

5.3.3.2 ADC_SWTEST_TEST2 (short detection)

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

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

During the first step the $V_{DD\_HV\_ADR\_SAR}$ is converted and compared with the expected value; then the $V_{SS\_HV\_ADR\_SAR}$ is converted and compared with the expected one (see Figure 5-6).

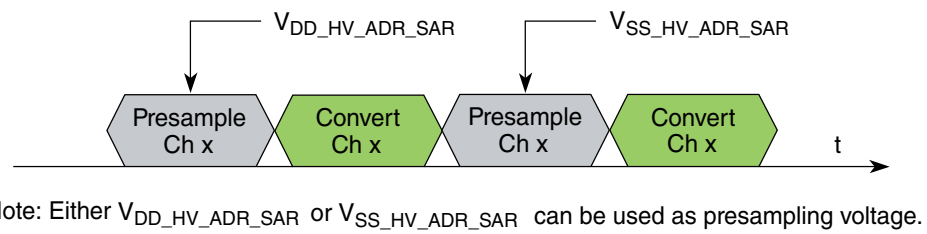


Figure 5-6. Implementation of ADC_SWTEST_TEST2

Rationale: To detect short failures of the channel multiplexing circuitry (see Figure 5-7).

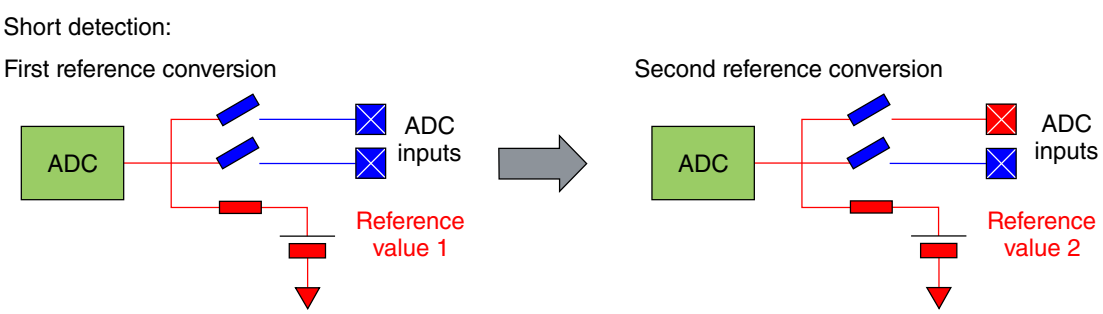


Figure 5-7. ADC_SWTEST_TEST2 (short detection)

Implementation hint: Presampling can be enabled on a per channel basis through the SARADC*n*_PSR0 register. SARADC*n*_PCSR[PREVAL0] selects which reference voltage is used to precharge/discharge the ADC internal capacitor. To bypass the

conversion of the input channel and convert the presampled values, SARADC*n*_PCSR[PRECONV] = 1. (See "Analog-to-Digital Converter (SARADC)" chapter in the MPC5746R Reference Manual for details on the presampling feature).

5.4 Communications

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

Assumption: It is assumed that communication over the FlexCAN interface will be protected by a fault-tolerant communication protocol. [end]

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

5.4.1 Redundant communication

Parts of the integrated DSPI, LINFlex, and SENT communication controller do not on their own provide the functional safety integrity ISO 26262 requires for high functional safety integrity targets. As these communication protocols often deal with low complex slave communication nodes, higher level functional safety protocols as described in Fault-tolerant communication protocol may not be feasible. Therefore, appropriate communication channel redundancy may be required. Multiple instances of communication controllers may be used to build up a single fault robust communication link.

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

- Synchronous Serial Communication Controller (DSPI)
- LINFlexD Communication Controller
- SENT

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

5.4.2 Fault-tolerant communication protocol

Parts of the integrated LINFlexD and FlexCAN communication channels do not on their own provide the functional safety integrity ISO 26262 requires for high functional safety integrity targets.

Assumption:[SM_200] Communication over the following interfaces shall be protected by a fault-tolerant communication protocol (implemented by the operating system or the application):[end]

- FlexCAN Communication Controller
- Universal Asynchronous Communication Controller (LINFlexD)
- Fast Ethernet Communication Controller (FEC)

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

Typically mechanisms are:

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

Assumption: [SM_300] For communication peripherals, ECC logic failures will be detected implicitly by the assumed FT-COM layer. [end]

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

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

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

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

5.5 Additional configuration information

5.5.1 Stack

Stack overflow and stack underflow is a common mode fault due to systematic faults within application software. A stack overflow occurs when using too much memory (pushing to much data) on the stack. A stack underflow occurs when software reads (pops) 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 (see section "Using Debug Resources for Stack Limit Checking" in the "Core Debug Support" chapter of the *MPC5746R Reference Manual*).

5.5.1.1 Initial checks and configurations

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

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

Implementation hint: Data Address Compare 1 (DAC1) and Data Address Compare 2 (DAC2) Special Purpose Registers (SPRs) may be used for incremental stack overflow or stack underflow detection when not being used as a hardware or software debug resource. Stack limit checking is available regardless of External Debug Mode (EDM) or Internal Debug Mode (IDM), and when resources used for stack limit checking are software controlled, will utilize a Data Storage Interrupt (DSI) or machine check exception. A data address compare (DAC) exception is signaled when there is a data access address match as defined by the debug control registers and data address compare events are enabled. This could either be a direct data address match or a selected set of data addresses, or a combination of data address and data value matching. The debug interrupt is taken when no higher priority exception is pending.

Software-owned stack limit checking does not require IDM to be set. Hardware owned stack limit checking requires EDM to be set. When stack limit checking is enabled, and DAC resources used for stack limit checking are owned by software, DAC events are not generated for resources configured to perform stack limit checking, and no DBSR DAC status flag will be set due to a detected stack limit violation. Instead, depending on the processor mode, a data storage interrupt or a machine check exception is signaled. When stack limit checking is enabled, and DAC resources used for stack limit checking are owned by hardware, DAC events will be generated for resources configured to perform stack limit checking, and the EDBSR0 DAC status flag will be set due to a detected stack limit violation, causing entry into debug halted mode in the same way as a DAC exception normally does. The only difference is that qualification of the access address is performed as discussed in the next paragraph.

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

When stack limit checking is enabled for a stack access, and DACn 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 andDAC2, access type (read, write) control is part of DBCR0.

5.5.2 MPC5746R configuration

Assumption:[SM_140] It is required that application software verifies that the initialization of the MPC5746R is correct before activating the safety-relevant functionality.[end]

After startup, the application software must ensure the conditions described in this section are satisfied before safety-relevant functions are enabled. Below is a list of the minimum number of checks by safety integrity functions which need to pass before executing any safety function:

- Lock-step mode check
- STCU check
- Flash Array Integrity Self check
- SUPPLY SELF-TEST
- Temperature sensor check
- SWT enabled
- CMU check
- IRC_SW_CHECK
- PMC check
- ERROR[n] signal check

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

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

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]

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 MPC5746R.

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 MPC5746R's safety function or disable the safety mechanism due to their change.

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

Rationale: To reduce the risk accidental writes configuration registers affecting the execution of the MPC5746R's safety function or disable the safety mechanism due to their change.

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

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

Chapter 6
Failure rates and FMEDA

6.1 Failure rates and FMEDA

6.1.1 Overview

According to ISO 26262-4, chapter 7.4.3.1, a functional safety/failure analysis on hardware design shall be applied to identify the causes of failures and the effects of faults. A typical inductive analysis method is FMEDA (Failure Modes Effects and Diagnostic Analysis).

Dedicated FMEDA and failure rate tables for ISO 26262 were created for each of the following parts of MPC5746R:

- FMEDAs for basic elements:
  - Core: processing units (CPU)
  - SRAM: non-volatile memories (SRAMs)
  - Flash: volatile memory (Flash)
  - Clock: clock generation and clock supervision
  - Power: Power generation and distribution
- Failure rates of application dependent functions:
  - I/O and peripherals

It is assumed, that the basic elements are used in every application and have low application dependency, whereas the use of peripheral and communications functions have a high application dependency. The functional safety architecture of basic elements may not interfere with the application.

The application dependent functions need to be included into the functional safety concept on system level. Thus only raw failure rates and no failure metrics are given for these elements.

The FMEDA enables selection of functional safety mechanisms planned to be implemented in a specific application. Enabling or disabling the use of functional safety mechanisms within an application is possible within the sheets.

The only failure modes used for the FMEDA are taken from table D.1 of ISO 26262-5, annex D. These are used for ISO 26262 calculations.

The implementation hints documented are assumed to be implemented as functional safety integrity measures.

6.1.2 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 MPC5746R 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 MPC5746R can be found in the attached "MPC5746R 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 MPC5746R on chip level:

- Random hardware failures, for example:
  - Physical defects that are able to influence an element and its redundant element.
  - Electrical dependencies:
    - Latch-up
    - Supply noise
    - Shared logic
    - Logic physically overlapping
    - Signals crossing lakes
    - Timing faults
- 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 (SEE, local overtemperature) whereas the synthesis achieves a partial diversity of the logic circuitry.

The master and checker core together with related logic, and the master and checker DMA 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 fatal and has to be detected by external system level measures. It is assumed that an external timeout function (watchdog) is continuously monitoring the MPC5746R and is capable of detecting this CCF, and will switch the system to a Safe $state_{system}$ within the FTTI.

The MPC5746R satisfies the standard AECQ100 for latch-up immunity.

7.1.2.2 Environmental Conditions

7.1.2.2.1 Temperature

MPC5746R was designed to work within a maximum operational temperature profile. For details, please refer to the MPC5746R Microcontroller Data Sheet. To cover CMFs cause by temperature, two temperature sensors for supervision are implemented which is described in 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 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 CMFs caused by clock, modules for supervision are implemented which are described in Clock Monitor Unit (CMU). Major failures in the clock system are also detected by the SWT (Software Watchdog Timer).

7.1.2.3.2 Power supply

To cover CMFs caused by issues with the power supplies, supervision modules are implemented (see Power Management Controller (PMC)). Some CMFs (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 Non-application control signals

Modules and signals (for example, for scan, test and debug), which are not functional safety-relevant and thus have no functional safety mechanism included should never be able to violate the functional safety goal. This can be achieved by either not interfering with the functional safety-relevant parts of the MPC5746R or by detecting such interference. For example, there must be assurance that the system is not debugged (or unintentionally in debug mode), or in any other special mode different from normal application execution mode like test mode. An FCCU failure indication is generated when one of the following conditions is fulfilled (see the "FCCU Fault Inputs" table in the Reference Manual for more details):

- The MPC5746R leaves LSM.
- A self-test sequence of the STCU is unintentionally executed during normal operation of the MPC5746R.
- Any of the configurations for production test are unintentionally executed during normal operation of the MPC5746R.
- 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 CMF 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 CMF. As internal pad position and external pin/ball position do not necessarily correspond to each other, the system integrator 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 common mode faults, measures on item (system) level have to improve the robustness of redundant inputs for double read input functions in respect to common mode faults.[end]

Recommendation: Avoid physically adjacent inputs for double read input functions to avoid CMFs.

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 *MPC5746R Microcontroller Data Sheet*. The following will explain how this can be achieved.

Recommendation: Redundant signals should have opposite polarity or use different protocols (time domain) to avoid dependency of signal inputs/outputs.

Recommendation: Time domain signal coding is preferred over state domain signal coding for safety critical signals.

7.1.3.1 I/O pin/ball configuration

Assumption:[SM_143] The user must avoid configurations that place redundant signals on neighboring pads or pins.[end]

Whether two functions on two signals are adjacent to each other can easily be determined by looking at the mechanical drawings of the packages (see the *MPC5746R Data Sheet*) together with the ball number information of the packages as seen in the *MPC5746R Reference Manuals* "System Integration Unit Lite (SIUL2)" section and the "Pin muxing" table (see also Table 7-1).

An example on the PBGA252 package as shown in Figure 7-1 has two balls belonging to port signals PH[4] and PH[5], which are balls D13 and A12, respectively. They are not directly adjacent to each other on the BGA package. However, their corresponding die pads are adjacent to each other as seen in Table 7-1 (die pads 175 and 176, respectively)

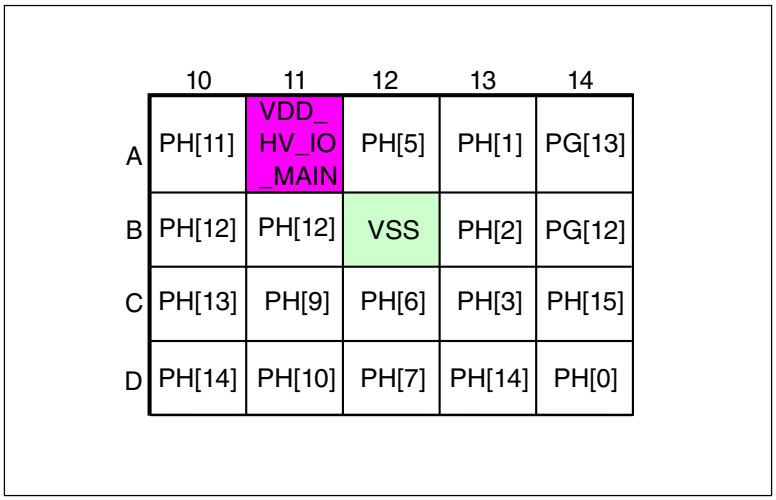


Figure 7-1. PBGA252 adjacency (partial view of package)

In another example looking balls C13 and D13 (port signals PH[3] and PH[4], respectively) you will notice that the balls are adjacent, but if you reference Table 7-1 you will also notice that the die pads are not adjacent (422 and 175, respectively). Therefore, the two corresponding die pads are not adjacent to each other.

The above examples are valid for corresponding balls on the BGA252. For a thorough analysis of pin adjacency related to all signals see Table 7-1. This table can be used to determine whether two pins are adjacent in the internal die for all signals and packages. Two pins, identified by the columns 'Port Name', are adjacent on the internal die if the numbers in the 'Physical Pad Sequence' column are consecutive (for example, pad number *n* and pad number *n* + 1 are adjacent).

Table 7-1. [SM_302] Physical pin displacement on internal die [end]

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PA[0]	A3	174	142	458
PA[1]	B3	173	141	457
PA[10]	A7	160	—	446
PA[11]	B7	159	—	445
PA[12]	B8	158	—	444
PA[13]	A8	157	—	443
PA[2]	A4	172	140	456
PA[3]	B4	171	139	455
PA[4]	C4	170	138	454
PA[5]	C5	169	137	453
PA[6]	B5	168	136	452
PA[7]	C6	165	—	449

Table continues on the next page...

Table 7-1. [SM_302] Physical pin displacement on internal die [end] (continued)

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PA[8]	C7	164	—	448
PA[9]	A6	161	—	447
PB[0]	D1	175	143	459
PB[1]	E3	176	144	460
PC[0]	Y9	56	45	299
PC[1]	W9	57	46	300
PC[10]	Y12	68	57	311
PC[11]	W12	69	58	312
PC[12]	V12	70	59	313
PC[13]	U12	71	60	314
PC[2]	V9	58	47	301
PC[3]	U9	59	48	302
PC[4]	W10	60	49	303
PC[5]	V10	61	50	304
PC[6]	U10	62	51	305
PC[7]	Y11	65	54	308
PC[8]	V11	66	55	309
PC[9]	U11	67	56	310
PD[0]	Y3	45	37	282
PD[1]	W3	46	38	283
PD[10]	W7	52	—	292
PD[11]	V7	53	—	294
PD[12]	U7	—	—	293
PD[13]	Y8	54	43	295
PD[14]	W8	55	44	297
PD[15]	V8	—	—	86
PD[2]	Y4	—	—	284
PD[3]	W4	—	—	75
PD[4]	V4	47	—	285
PD[5]	Y5	—	—	286
PD[6]	W5	—	—	288
PD[7]	V5	48	39	287
PD[8]	V6	49	40	289
PD[9]	Y7	—	—	82
PE[0]	U8	—	—	87
PF[0]	G20	126	—	386
PF[1]	G19	127	—	387
PF[10]	D18	—	—	401
PF[11]	C20	130	106	402

Table continues on the next page...

Table 7-1. [SM_302] Physical pin displacement on internal die [end] (continued)

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PF[12]	C19	131	107	405
PF[13]	B20	132	108	406
PF[2]	G18	—	—	149
PF[3]	G17	—	—	389
PF[4]	F19	—	—	390
PF[5]	F18	—	—	391
PF[6]	E20	—	—	393
PF[7]	E18	—	—	394
PF[8]	D20	—	—	396
PF[9]	D19	—	—	400
PG[1]	A18	133	109	407
PG[10]	A15	138	—	412
PG[11]	B15	140	115	414
PG[12]	B14	141	116	415
PG[13]	A14	142	117	416
PG[14]	C15	—	—	169
PG[15]	C14	—	—	171
PG[2]	A17	134	110	408
PG[3]	B17	—	—	162
PG[4]	C18	—	—	404
PG[5]	B16	135	111	409
PG[6]	A16	136	112	410
PG[7]	C17	—	—	403
PG[9]	C16	137	113	411
PH[0]	D14	143	118	417
PH[1]	A13	—	—	172
PH[10]	D11	—	—	182
PH[11]	A10	150	125	427
PH[12]	B10	151	126	428
PH[13]	C10	—	—	185
PH[14]	D10	152	127	429
PH[15]	A9	153	128	436
PH[2]	B13	—	—	413
PH[3]	C13	144	119	422
PH[4]	D13	—	—	175
PH[5]	A12	—	—	176
PH[6]	C12	—	—	180
PH[7]	D12	145	120	423
PH[8]	B11	146	121	424

Table continues on the next page...

Table 7-1. [SM_302] Physical pin displacement on internal die [end] (continued)

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PH[9]	C11	—	—	181
PI[0]	B9	154	129	438
PI[1]	C9	155	130	439
PI[2]	D9	—	—	192
PI[3]	C8	156	131	440
PI[4]	D8	—	—	437
PI[5]	D7	—	—	191
PJ[0]	H1	10	10	231
PJ[1]	G4	11	11	232
PJ[10]	K4	—	—	49
PJ[11]	L3	25	24	256
PJ[12]	L4	26	—	257
PJ[13]	M3	27	—	259
PJ[14]	M4	28	25	260
PJ[15]	N2	29	—	261
PJ[2]	H2	—	—	229
PJ[3]	H3	12	—	233
PJ[4]	H4	13	12	234
PJ[5]	J2	18	17	244
PJ[6]	J3	—	—	31
PJ[7]	J4	19	18	245
PJ[8]	K2	—	—	46
PJ[9]	K3	—	—	48
PK[0]	N3	30	—	263
PK[1]	N4	31	26	264
PK[10]	T2	40	33	277
PK[11]	T3	41	34	278
PK[12]	U1	42	—	279
PK[13]	U2	43	35	280
PK[14]	V1	44	36	281
PK[2]	P1	32	27	265
PK[4]	P2	33	—	266
PK[5]	P3	34	28	267
PK[7]	P4	37	31	271
PK[8]	R1	38	32	273
PK[9]	R3	39	—	275
PORST	G3	9	9	228
PW[0]	Y13	—	—	316
PW[1]	W13	76	64	318

Table continues on the next page...

Table 7-1. [SM_302] Physical pin displacement on internal die [end] (continued)

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PW[2]	V13	75	—	317
PW[3]	U13	74	63	315
PX[0]	U19	—	—	336
PX[1]	U18	89	73	334
PX[10]	W15	81	66	324
PX[11]	V15	—	—	112
PX[12]	Y14	80	—	323
PX[13]	W14	79	—	322
PX[14]	V14	78	65	321
PX[15]	V16	77	—	320
PX[2]	V18	88	72	333
PX[3]	Y17	87	71	332
PX[4]	W17	—	—	118
PX[5]	V17	84	68	327
PX[6]	Y16	—	—	119
PX[7]	W16	83	67	326
PX[8]	U14	—	—	319
PX[9]	Y15	82	—	325
PY[0]	N20	101	—	357
PY[1]	N19	100	80	356
PY[10]	R18	94	—	347
PY[11]	T20	—	—	346
PY[12]	T19	—	—	345
PY[13]	T18	93	77	344
PY[14]	U20	—	—	343
PY[15]	V20	90	74	335
PY[2]	N18	99	—	355
PY[3]	N17	—	—	354
PY[4]	P20	98	—	353
PY[5]	P19	—	—	352
PY[6]	P18	97	79	351
PY[7]	P17	96	—	350
PY[8]	R20	—	—	349
PY[9]	R19	95	78	348
PZ[0]	H20	123	102	384
PZ[1]	H19	122	101	383
PZ[10]	L18	109	88	364
PZ[11]	L17	108	87	363
PZ[12]	M18	107	86	362

Table continues on the next page...

Table 7-1. [SM_302] Physical pin displacement on internal die [end] (continued)

Port name	Ball number PBGA252	Pin number QFP176	Pin number QFP144	Physical pad sequence[1]
PZ[13]	M17	106	85	361
PZ[14]	M20	105	84	360
PZ[15]	M19	104	83	359
PZ[2]	H18	121	100	382
PZ[3]	H17	120	99	381
PZ[4]	J20	119	98	380
PZ[5]	J19	118	97	379
PZ[6]	J18	117	96	378
PZ[7]	J17	116	95	377
PZ[8]	K18	111	90	366
PZ[9]	K17	110	89	365
—	—	—	—	—

1. Die pads not relevant for analysis, and non-functional pins (for example, power) are not shown.

7.1.3.2 Modules sharing PBRIDGE

The system designer needs to take into consideration how modules are distributed across the different PBRIDGEs. Whenever possible the redundant modules should be used such that each module is connected to a different PBRIDGE. For example, SARADC_0 and SARADC_2 are on PBRIDGEA while SARADC_1 and SARADC_3 connect to PBRIDGEB. So, when redundancy is required for the safety function, the designer should utilize SARADC_0 or SARADC_2 with either SARADC_1 or SARADC_3.

7.1.3.3 External timeout function

A CMF may lead to a state where the MPC5746R 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 CMFs affect the functional safety of the system can be reduced significantly.

In general, the external watchdog covers CMFs 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 MPC5746R 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, especially unexpected errors (for example, test, debug, sleep/ wakeup)
    - 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 CCFs with wider than expected effects (for example, disabling XOSC and IRCOSC at the same time).

Since these errors do not result in subtle output variations of the MPC5746R but typically in a complete failure, a simple watchdog is sufficient.

The external watchdog function is in permanent communication with the CPU of MPC5746R. As soon as there are no correct communications, the external watchdog function switches the system to Safe $state_{system}$. Thus, either the MPC5746R or external watchdog function can transition the system to Safe $state_{system}$. The external watchdog function is required to be sufficiently independent of the MPC5746R (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 (for example, via CAN) and moving the system to Safe $state_{system}$.

Chapter 8
Additional information

8.1 Additional information

8.1.1 Checks and configurations

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

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

- Lock-step mode check
- STCU check
- Flash Array Integrity Self check
- SUPPLY SELF-TEST
- Temperature sensor check
- SWT enabled
- CMU check
- IRC_SW_CHECK
- PMC check
- ERROR[*n*] signal check[1]

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

1. Required for single FCCU signal usage only

8.2 Testing All-X in RAM

As mentioned in section "End-to-end ECC", 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 provides 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 anyway by an application show this desired behaviour.

8.2.1 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`

8.2.2 ECC checkbit/syndrome coding scheme

The e2eECC scheme implements a SECDED code using the 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 a 64-bit data field for the computational shell (32-bit data field for the peripheral shell) plus 29-bit address field (such as, the upper bits of the 32-bit address field minus the 3 bits which select the byte within the 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. E2E ECC basic H-matrix definition

Checkbits [7:0]	Data Bit																															
	Byte 7								Byte 6								Byte 5								Byte 4							
	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
7	*				*					*			*		*	*		*	*	*	*					*	*			*	*	
6			*	*			*	*					*		*		*				*				*			*	*			*
5	*	*	*	*					*				*	*		*		*	*				*	*	*				*			

Table continues on the next page...

Table 8-1. E2E ECC basic H-matrix definition (continued)

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 E2E ECC logic performs the same type of optional adjustment on the read checkbits.

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

Chapter 9
Acronyms and abbreviations

9.1 Acronyms and abbreviations

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

Table 9-1. Acronyms and abbreviations

Terms	Meanings
ADC	Analog to Digital Converter
BAM	Boot Assist Module
CCF	Common Cause Failure
CMF	Common Mode Failures
CMU	Clock Monitor Unit
CRC	Cyclic Redundancy Check
CTU	Cross-Triggering Unit
DC	Diagnostic Coverage
ECC	Error Correcting Code
ECSM	Error Correction Status Module
eDMA	Direct Memory Access
ERRM	Error Out Monitor function
EXWD	External Watchdog function
FCCU	Fault Collection and Control Unit
FMEDA	Failure Modes, Effects & Diagnostic Analysis
FMPLL	Frequency-Modulated Phase-Locked Loop
GPIO	General Purpose Input/Output
LBIST	Logic Built-In Self Test
MBIST	Memory Built-In Self Test
MC_CGM	Clock Generation Module
MC_ME	Mode Entry
MC_RGM	Reset Generation Module
MCU	Microcontroller Unit

Table continues on the next page...

Table 9-1. Acronyms and abbreviations (continued)

Terms	Meanings
MPU	Memory Protection Unit
NCF	Non-Critical Fault
NMI	Non-Maskable Interrupt
NVM	Non-Volatile Memory
PMU	Power Management Unit
PSM	Power Supply and Monitor function
PWM	Pulse Width Modulation
RCCU	Redundancy Control Checking Unit
SIL	Safety Integrity Level
SM	Safety Manual
SWT	Software Watchdog Timer

Appendix A
Release Notes for Revision 2.1

A.1 General changes

- Rev. 2.1 removes the confidential-proprietary and preliminary footers and 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.

A.2 Preface changes

- In the Preface section:
  - Changed "ASIL-D applications" to "safety-related applications".
  - Changed "ISO/DIS 26262-10" to "ISO 26262-10".

A.3 General Information changes

- In the Safety function section:
  - Replaced the previous "Safety function" content with the new "MCU Safety function" subsection.
  - Moved the Correct operation section here as a subsection.
- In the Mission Profile section:
  - Removed the "Temperature Profile" table and associated content.
- In the Correct operation section:
  - Replaced this entire section. The module classification xls file (attached to this document) replaces the "Correct Operation of System Modules" table that formerly appeared here.
  - Moved this section to the Safety function section.

- In the Mission Profile section:
  - Removed the assumption number from the "Fault-Tolerant Time Interval (FTTI)" bullet.
- In the Single-point Fault Tolerant Time Interval section:
  - Changed TBDs to actual values:
    - For "Fault detection time", changed "ADC recognition time" from TBD to 70 µs.
    - For "Fault detection time", changed "Recognition time related to the FMPLL loss of clock" from TBD to 20 µs.
    - For "Fault reaction time", "External indication time", changed "FCCU configured as fast switching mode indication delay" from TBD to 64 µs.

- For "Fault reaction time", "External indication time", changed "FCCU configured as slow switching mode indication delay" from TBD to 8 ms.
- For "Fault reaction time", "External indication time", changed "Bistable protocol indication delay" from TBD to 64 µs.
- Updated the "FCCU configured as slow switching mode" description.

A.4 Functional Safety Concept changes

- In the General functional safety concept section:
  - In the "Replication of core processing elements" list, changed "Core Local Data Memory (D-MEM)" to "Core Local Memory Controller".
  - In the "Error correction or detection to reduce the effect of faults" list, added bullets "System RAM", "Overlay RAM", "I-MEM", and "eTPU SCM and SDM".

A.5 Hardware Requirements changes

- In the High impedance outputs section:
  - Removed the paragraph regarding "System level countermeasures".
  - Removed the words "Implementation hint" but retained the related content.
- In the External Watchdog (EXWD) section:
  - Replaced this section.
- In the Power Supply Monitor (PSM) section:
  - Added SM_086, SM_087, and SM_088.
  - Changed "Over-voltage on the 1.25 V core supply will be detected" to "Over-voltage on the 1.25 V core supply can be detected".
  - Added the condition that the above over-voltage will be detected "if the core supply LVD has been enabled by properly setting the related DCF record".
  - Added the paragraph "Some internal voltage monitors can be enabled or disabled..."
- In the Error Out Monitor (ERRM) section:
  - Removed the statement "Both FCCU configurations work properly with all supported error out protocols..."
  - Removed the Recommendation.
- In the Both FCCU signals connected to separate device section:
  - Changed "Implementation hint: If both ERROR[0] and ERROR[1]..." to be an Assumption.
  - In the same sentence, changed "If both ERROR[0] and ERROR[1] are connected" to "If both error out signals are connected".
  - In the same sentence, changed "the device may check" to "the external device shall check".
  - In the same sentence, changed "ERROR[0]=ERROR[1]" to "the behavior of the two pins".
  - Changed the paragraph "Monitoring the ERROR[0] and ERROR[1] through asynchronous combinatorial logic..." to be an Implementation Hint.
- In the Single FCCU signal connected to separate device using voltage domain coding section:
  - Updated the paragraph "If the system is using the MPC5746R... " as follows:
    - Changed the first sentence to be an Assumption.
    - Added to the first sentence: "and the error output signal is configured with highest drive strength".
    - Editorial improvements.
- In the Single FCCU signal line connected to separate device using time domain coding section:
  - Removed the paragraph "A time domain coding of ERROR[0]..."
  - Removed the sentence "It is recommended to toggle ERROR[0]..."
- In the PowerSBC section:
  - Added this new section.

- In the Assumed functions by separate circuitry section:
  - Changed "ASIL-D applications" to "safety-related applications".

A.6 Software Requirements changes

- In the Fault Collection and Control Unit (FCCU) section:
  - Removed the "FCCU mapping of faults" table.
  - Added text referring the reader to the "FCCU Fault Inputs" table in the Reference Manual.
- In the FCCU Runtime Checks section:
  - Added the Implementation hint "Before the safety application clears the reset counters..."
- In the STCU2 Initial checks and configurations section:
  - Added the Implementation hint "The integrity software shall confirm that all MBISTs and LBISTs finished successfully with no additional errors flagged."
- In the Temperature Sensors (TSENS) section:
  - Removed the text "TSENS1 (is read) from SARADC3 channel 45, TSENS2 (is read) from SARADC3 channel 47".
  - Removed SM_275.
- In the RCCU Initial checks and configurations section:
  - Added the paragraph "However, LSM can be disabled during boot by reprogramming the flash memory..."
- In the IRCOSC Runtime Checks section:
  - Added the Implementation hint.
- In the External Oscillator (XOSC) section:
  - Removed "For applications targeting ASIL D, this clocking mode should not be used."
- In the XOSC Initial checks and configurations section:
  - Replaced Assumption SM_075 with "FlexCAN should not be clocked directly by the XOSC in normal operation unless the effects of clock glitches are sufficiently detected by the applied FT-COM layer."
- In the Dual PLL Digital Interface (PLLDIG) section:
  - Removed "NCF[32], NCF[33]" from SM_077.
- In the PLLDIG Initial checks and configurations section:
  - Removed the text "During/after initialization but before executing any safety function..."
  - Removed the Implementation hint "Application software can check the current system clock..."
  - Added the Implementation hint "Either during or after initialization..."
- In the Clock Monitor Unit (CMU) section:
  - Removed "Clocks are supervised by Clock Monitoring Units (CMUs)."
  - Replaced "see Table 5-1, "FCCU mapping of faults", for details" with "please see the table "FCCU Fault Inputs" in the Reference Manual for details".
- In the Power Management Controller (PMC) section:
  - Removed the list of monitored voltages.
  - Added Assumption SM_204.
  - Removed Assumption SM_086.
  - Removed Assumption SM_087.
  - Removed Assumption SM_088.
  - Updates to the "PMC monitored supplies" table:
    - Changed LVD_VDDFLASH to LVD_FLASH.
    - Added HVD_FLASH.
    - Removed the following: VDD_HV_IO_JTAG, VDD_HV_IO_FEC, VDD_HV_IO_MSC, LVD_JTAG, LVD_FEC, LVD_MSC_3V3, LVD_MSC_5V0, 5.0 V JTAG supply, 5.0 V FEC supply, 3.3 V OR 5.0 V MSC supply, VDD_HV_ADV_SD, LVD_SD, HVD_SD, 5.0 V Sigma Delta ADC supply, HVD_SAR.
- In the 1.25 V supply supervision section:
  - In the "Logic scheme of the core voltage detectors" figure:
    - Changed "HVD_VDD" to "HVD_core".
    - Changed "LVD_VDD" to "LVD_core_hot/cold".
- In the 3.3 V supply supervision section:
  - Removed LVD_MSC_3V3, LVD_FEC, and LVD_JTAG from the listed voltage detectors.
  - Specified that the 3.3 V supply is the internal 3.3 V regulator.
- In the SMPU Initial checks and configurations section:
  - In the first Rationale, added that access restriction:
    - Is at the MPU level.
    - Applies to specific software routines.

- In the Implementation hint, removed the list of bus masters.
- Removed the second (redundant) Rationale.
- Removed the "Recommended" paragraph.

- In the Built-in Hardware Self-Tests (BIST) section:
  - Removed subsection "Flash memory ECC logic check".
  - Removed subsection "Flash memory ECC fault report check".
- In the End-to-end ECC (e2eECC) section:
  - In the "Single Error Correction/Double Error Detection" bullet, specified that coverage is "for the computational shell (32-bit data field for the peripheral shell)".
- In the Periodic low latency IRQs section:
  - Changed "The SWT can be configured..." to "The Interrupt Control Monitor (INTCM) can be configured...".
- In the SSCM Initial checks and configurations section:
  - Changed all occcurrences of "BAM" to "BAF" (Boot Assist Flash).
- In the Memory Error Management Unit (MEMU) section:
  - Added this paragraph: "All errors that the MEMU collects are stored in reporting tables that are accessible through the MEMU register interface."
- In the Flash Runtime Checks section:
  - Completely revised the second Implementation hint.
- In the Wake-Up Unit (WKPU) / External NMI section:
  - Removed mention of the analog filter in the Rationale.
- In the Crossbar Switch (XBAR) section:
  - Removed the short lists of masters and slaves from the first paragraph.
  - Removed Assumptions SM_083, SM_289, and SM_299.
- In the I/O functions section:
  - Added the two Implementation hints "Possible measures..."
- In the Communications section:
  - Added the content to this (previously empty) section.
- In the Stack Initial checks and configurations section:
  - Created this new subsection.
- In the MPC5746R configuration section:
  - Added the list of minimum checks needed before executing any safety function and accompanying text.

- In the Self Test Control Unit (STCU2) section:
  - Removed MC_RGM as an option for receiving signaling of faults from the STCU2.
- In the STCU2 Initial checks and configurations section:
  - In the last "Implementation hint", changed "test flash memory" to "UTest flash memory".
- In the TSENS Initial checks and configurations section:
  - In the first "Assumption", changed the "see Table 5-1" reference that pointed to the "FCCU mapping of faults" table to "see the "FCCU Fault Inputs" table in the Reference Manual".
  - In the last "Assumption", removed the reference to SARADC3 channels 45/47.
- In the Software Watchdog Timer section:
  - Changed the "see Table 5-1" reference that pointed to the "FCCU mapping of faults" table to "see the "FCCU Fault Inputs" table in the Reference Manual".
  - Changed "ASIL D applications" to "safety-related applications".
- In the Clock Monitor Unit (CMU) section:
  - Changed "ASIL D applications" to "safety-related applications".
- In the Power Management Controller (PMC) section:
  - Changed "ASIL D applications" to "safety-related applications".
  - Changed the word "powerless" to "unpowered".
  - Added after the table: "Additionally, the SD ADC supply and reference and IO segment supplies can be monitored using the SAR ADC. See the MPC5746R Reference Manual for further details."
- In the Memory Protection Units section:
  - Changed "ASIL D applications" to "safety-related applications".
- In the Core Memory Protection Unit (CMPU) section:
  - Changed "ASIL D applications" to "safety-related applications".
- In the System Timer Module (STM) section:
  - Editorial (non-technical) change.
- In the Periodic Interrupt TImer (PIT) section:
  - Editorial (non-technical) change.
- In the System Status and Control Module (SSCM) section:

- Editorial (non-technical) change.
- Changed "non-ASIL D" to "non-safety-related application".

- In the Body Cross Triggering Unit (BCTU) section:
  - Editorial (non-technical) change.
- In the Error reporting path tests section:
  - Changed "The FCCU input table specifically lists those inputs in the 'Suggested fault reaction' column in Table 5-1" to "See the "FCCU Fault Inputs" table in the Reference Manual for more details".
  - Removed the "Assumption" and "Rationale".
- In the Register Protection module (REG_PROT) section:
  - Changed "ASIL D applications" to "safety-related applications".
- In the Crossbar Switch (XBAR) section:
  - Changed "ViMos or SuMos" to "safety-related modules".

- In the MPC5746R modules section:
  - Removed the content of this section because it was a duplicate of the Fault-tolerant communication protocol section.
- In the Flash Runtime checks section:
  - Changed the Assumption "A software safety mechanism shall be implemented to ensure the correctness of any write operation to the flash memory" to include the overlay.
  - Changed the Assumption "Flash memory ECC failure reporting has to be executed within the FTTI to validate if detected ECC faults are communicated properly to the FCCU and other necessary modules" to "The Flash memory ECC failure reporting path should be checked to validate if detected ECC faults are correctly reported."
- In the ADC Initial checks and configurations section:
  - Replaced the "Mapping of test flash memory values to STAWxR" table with the "Sample Values for ADC Self-Test Thresholds" table.
- In the Stack Initial checks and configurations section:
  - Added the Assumption number to the "Assumption under certain conditions".

A.7 Failure Rates and FMEDA changes

- In the Failure rates and FMEDA section:
  - Removed the "Note" that stated the FMEDA was incomplete.
- In the Overview section:
  - Removed the "Module distribution over FMEDAs" table and associated content.
  - Removed the list of modules that are covered by the failure rates list.
  - Removed the paragraph "The information given in this section is valid as of the date of generation of this document..."
  - Removed the paragraph "Significant key values of the FMEDA are presented in a FMEDA report document..."
  - Removed the paragraph "The failure rate data used in these FMEDAs have been derived..."
- Added the new section Module classification.

A.8 Dependent Failures changes

- In the Modules sharing PBRIDGE section:
  - Corrected "SARADC1" to "SARADC_1".

- In the Non-application control signals section:
  - Changed the reference that pointed to Table 5-1 "FCCU mapping of faults" to "see the "FCCU Fault Inputs" table in the Reference Manual".

A.9 Additional Information changes

- No substantial content changes

A.10 Acronyms and Abbreviations changes

- No substantial content changes

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MPC5746RSM
Revision 2.1, 07/2017