Safety Manual for MPC5748G

Devices Supported: MPC5748G

Document Number: MPC5748GSM
Rev. 3, 08/2017

Contents

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

Chapter 1
Preface

1.1 Overview 11

1.2 Safety manual assumptions 11

1.3 Safety manual guidelines 12

1.4 Functional safety standards 12

1.5 Related documentation 13

1.6 Other considerations 13

Chapter 2
MCU Safety Context

2.1 Target Applications 15

2.2 Safety integrity level 15

2.3 Safety function 15

2.3.1 MCU safety functions 15

2.3.2 Correct operation 16

2.4 Safe states 17

2.4.1 MCU Safe state 17

2.4.2 Transitions to Safe statesystem 18

2.4.3 Continuous reset transitions 19

2.5 Faults and failures 19

2.5.1 Failure types 19

2.5.2 Faults 19

2.5.3 Dependent failures 21

2.6 Single-point fault tolerant time interval and process safety time 23

2.6.1 MCU fault indication time 24

2.7 Latent-fault tolerant time interval for latent faults 25

2.7.1 MCU fault indication time 26

2.8 MCU failure indication 27

2.8.1 Failure handling....27
2.8.2 Failure indication signaling....27

Chapter 3
MCU Safety Concept

3.1 General concept....29
3.2 Use of cores for safety - self-test, reciprocal comparison, temporal redundancy....31
3.3 ECC....31
3.3.1 End-to-End protection on data path....31
3.3.2 ECC for storage....33
3.3.3 All-X words and ECC....34
3.3.4 ECC failure handling....34
3.4 Clock and power monitoring....35
3.4.1 Clock....35
3.4.2 Power....35
3.5 I/O peripherals....36
3.6 Communication controllers....36
3.6.1 Disabling of communication controllers....36
3.7 Built-In Self Tests (BIST)....37
3.7.1 BIST during boot....38
3.7.2 LBISTed modules....38
3.8 FCCU and failure monitoring....40
3.8.1 External error indication....40
3.8.2 Failure handling....41
3.8.3 Fault inputs....41
3.9 Memory Error Management Unit (MEMU)....42
3.9.1 Interface to ECC units....42
3.10 Operational interference protection....43
3.11 Common cause failure measures....44

Chapter 4

Hardware Requirements

4.1 Hardware requirements on system level....45
4.1.1 Assumed functions by separate circuitry....46
4.1.1.1 High impedance outputs....46
4.1.1.2 External Watchdog (EXWD)....46
4.1.1.3 Power Supply Monitor (PSM)....47
4.1.1.4 Error Out Monitor (ERRM)....48
4.1.2 Optional hardware measures on system level....51
4.1.2.1 External communication....51
4.1.2.2 PWM output monitor....51

Chapter 5
Software Requirements

5.1 Software requirements on system level....53
5.1.1 Disabled modes of operation....53
5.1.1.1 Debug mode....53
5.1.1.2 Test mode....54
5.2 MPC5748G modules....54
5.2.1 Cores....55
5.2.1.1 Runtime checks....55
5.2.2 Fault Collection and Control Unit (FCCU)....57
5.2.2.1 Initial checks and configurations....58
5.2.2.2 Runtime checks....59
5.2.3 Reset Generation Module (MC_RGM)....60
5.2.3.1 Initial checks and configurations....60
5.2.4 Self Test Control Unit (STCU2)....61
5.2.4.1 Initial checks and configurations....61
5.2.5 Software Watchdog Timer....62
5.2.5.1 Run-time checks....63
5.2.6 Cyclic Redundancy Checker Unit....64

5.2.6.1 Runtime checks....64
5.2.7 Slow Internal RC Oscillator....67
5.2.8 Fast Internal RC Oscillator (FIRC)....67
5.2.8.1 Initial checks and configurations....67
5.2.8.2 Runtime checks....68
5.2.9 Fast External Oscillator (FXOSC)....68
5.2.9.1 Initial checks and configurations....68
5.2.9.2 Runtime checks....69
5.2.10 PLL Digital Interface (PLLDIG)....69
5.2.10.1 Initial checks and configurations....69
5.2.11 Clock Monitor Unit (CMU)....70
5.2.11.1 Initial checks and configurations....71
5.2.12 Mode Entry (MC_ME)....71
5.2.13 Power Management Controller (PMC)....72
5.2.13.1 1.25 V supply supervision....73
5.2.13.2 3.3 V supply supervision....73
5.2.13.3 5 V supply supervision....74
5.2.14 Memory Protection Units....74
5.2.14.1 System Memory Protection Unit (SMPU)....74
5.2.14.2 Initial checks and configurations....75
5.2.15 Peripheral Bridge (PBRIDGE) protection....76
5.2.15.1 Initial checks and configurations....76
5.2.16 Built-in Hardware Self-Tests (BIST)....76
5.2.16.1 Memory Built-In Self-Test (MBIST)....78
5.2.16.2 Logic Built-In Self-Test (LBIST)....78
5.2.16.3 Flash memory array integrity self check....79
5.2.16.4 Flash memory ECC logic check....79
5.2.16.5 Flash memory ECC fault report check....79
5.2.17 End-to-end ECC (e2eECC)....79

5.2.18 Interrupt Controller (INTC) 80
5.2.18.1 Periodic low latency IRQs 81
5.2.18.2 Non-Periodic low latency IRQs 81
5.2.18.3 Runtime checks 81
5.2.19 Enhanced Direct Memory Access (eDMA) 81
5.2.19.1 Runtime checks 82
5.2.20 System timer module 83
5.2.20.1 Runtime checks 83
5.2.21 Periodic interrupt timer 83
5.2.21.1 Runtime checks 83
5.2.22 System Status and Configuration Module 84
5.2.22.1 Initial checks and configurations 84
5.2.23 Memory Error Management Unit (MEMU) 84
5.2.24 Flash memory 84
5.2.24.1 EEPROM 85
5.2.24.2 Initial checks and configurations 85
5.2.24.3 Runtime checks 85
5.2.25 Body Cross Triggering Unit (BCTU) 86
5.2.25.1 Runtime checks 87
5.2.25.2 Synchronize sequentially read inputs 87
5.2.26 Error reporting path tests 88
5.2.27 Glitch filter 89
5.2.28 Register Protection module (REG_PROT) 89
5.2.28.1 Runtime checks 90
5.2.29 Wake-Up Unit (WKPU) / External NMI 90
5.2.30 Crossbar Switch (AXBS) 91
5.2.30.1 Runtime checks 91
5.2.31 System Integration Unit Lite2 (SIUL2) 91
5.2.31.1 Digital inputs 92

5.2.31.2 Hardware....92
5.2.32 Analog-to-Digital Converter (ADC)....92
5.2.32.1 Initial checks and configurations....93
5.3 Communications....93
5.3.1 Redundant communication....93
5.3.2 Fault-tolerant communication protocol....94
5.4 Additional configuration information....95
5.4.1 Stack....95
5.4.1.1 Initial checks and configurations....95
5.4.2 MPC5748G configuration....97

Chapter 6
Failure Rates and FMEDA

6.1 Failure rates....99
6.2 FMEDA....99
6.2.1 Module classification....100

Chapter 7
Dependent Failures

7.1 Provisions against dependent failures....101
7.1.1 Causes of dependent failures....101
7.1.2 Measures against dependent failures....102
7.1.2.1 Environmental conditions....102
7.1.2.2 Failures of common signals....102
7.1.3 Dependent failure avoidance on system level....103
7.1.3.1 I/O pin/ball configuration....103
7.1.3.2 Modules sharing PBRIDGE....104
7.1.3.3 External timeout function....104
7.1.4 βIC considerations....105

Chapter 8
Additional Information

8.1 Testing All-X in RAM....107

8.1.1 Candidate address for testing All-X issue..... 107

8.1.2 ECC checkbit/syndrome coding scheme..... 112

Chapter 9
Acronyms and Abbreviations

9.1 Acronyms and abbreviations..... 117

Chapter 1
Preface

1.1 Overview

This document discusses requirements for the integration and use of the MPC5748G Microcontroller Unit (MCU) in safety-related systems. It is intended to support safety system developers in building their safety-related systems using the safety mechanisms of the MPC5748G, and describes the system level hardware or software safety measures that should be implemented to achieve the desired system level functional safety integrity level. The MPC5748G is developed according to ISO 26262 and has an integrated safety concept.

1.2 Safety manual assumptions

During the development of the MPC5748G, assumptions were made on the system level safety requirements with regards to the MCU. During the system level development, the safety system developer is required to establish the validity of the MCU assumptions in the context of the specific safety-related system. To enable this, all relevant MCU assumptions are published in the Safety Manual and can be identified as follows:

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

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

Example: **Assumption under certain conditions:** If a high impedance state on an output is not safe, pull-up or pull-down resistors shall be added to safety-critical outputs. The need for this will be application dependent for the unpowered or reset condition (tristated I/O) of the MPC5748G.

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

1.3 Safety manual guidelines

This document also contains guidelines on how to configure and operate the MPC5748G in safety-related systems. These guidelines are preceded by one of the following text statements:

- **Recommendation:** A recommendation is either a proposal for the implementation of an assumption, or a reasonable measure which is recommended to be applied, if there is no assumption in place. The safety system developer has the choice whether or not to adhere to the recommendation.
- **Rationale:** The motivation for a specific assumption and/or recommendation.
- **Implementation hint:** An implementation hint gives specific details on the implementation of an assumption and/or recommendation on the MPC5748G. The safety system developer has an option to follow the implementation hint.

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

1.4 Functional safety standards

It is assumed that the user of this document is familiar with the functional safety standards *ISO 26262 Road vehicles - Functional safety* and *IEC 61508 Functional safety of electrical/electronic/programmable electronic safety-related systems.* The MPC5748G

is a component as seen in the context of ISO 26262 and in this case its development is completely decoupled from the development of an item or system. Therefore the development of the MPC5748G is considered a Safety Element out of Context (SEooC) development, as described in *ISO 26262-10.9 Safety element out of context* and more specifically detailed in *ISO 26262-10.9.2.3 Development of a hardware component as a safety element out of context* and *ISO 26262-10:2011-2012 Annex A ISO 26262 and microcontrollers*.

1.5 Related documentation

The MPC5748G is developed according to ISO 26262 and has an integrated safety concept targeting safety-related systems requiring high safety integrity levels. In order to support the integration of the MPC5748G into safety-related systems, the following documentation will be available:

- Reference Manual (MPC5748GRM) - Describes the MPC5748G functionality
- Data Sheet (MPC5748GDS) - Describes the MPC5748G operating conditions
- Safety Manual (MPC5748GSM) - Describes the MPC5748G safety concept and possible safety mechanisms (integrated in MPC5748G, system level hardware or system level software), as well as measures to reduce dependent failures
- FMEDA - Inductive analysis enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and ß-factor $ß_{IC}$)
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request. The MPC5748G is a SafeAssure solution; for further information regarding functional safety at NXP, visit www.nxp.com/safeassure.

1.6 Other considerations

When developing a safety-related system using the MPC5748G, the following information should be considered:

- The MPC5748G is handled in accordance with JEDEC standards J-STD-020 and J-STD-033.
- The operating conditions given in the MPC5748G Data Sheet.
- If applicable, any published MPC5748G errata.

- The recommended production conditions given in the MPC5748G quality agreement.
- The functional safety manager for the developed and deployed system is required to report all field failures of the MPC5748G to NXP.

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

Chapter 2
MCU Safety Context

2.1 Target Applications

The family of devices are designed to address a wide variety of automotive applications including but not limited to the door modules, seat modules, central body, vehicle body controllers, smart junction box, front module applications, high end gateway or combined body controller and gateway applications.

2.2 Safety integrity level

The MPC5748G is designed to be used in automotive, or industrial, applications which need to fulfill functional safety requirements as defined by functional safety integrity levels (for example, ASIL B of ISO 26262 or SIL 2 of IEC 61508). The MPC5748G is a component as seen in the context of ISO 26262 and in this case its development is completely decoupled from the development of an item or system. Therefore the development of the MPC5748G is considered a Safety Element out of Context (SEooC) development.

The MPC5748G is seen as a Type B subsystem in the context of IEC 61508 (“complex,” see IEC 61508-2, section 7.4.4.1.3) with a HFT = 0 (Hardware Fault Tolerance) and may be used in any mode of operation (see IEC 61508-4, section 3.5.16).

2.3 Safety function

2.3.1 MCU safety functions

Given the application independent nature of the MPC5748G, no specific safety function can be specified. Therefore, during the SEooC development of the MPC5748G, 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 MPC5748G flash memory, buffer these within instruction cache, execute instructions, read data from the MPC5748G System SRAM or flash memory, buffer these in data cache, process data and write result data into MPC5748G System SRAM. **Functional safety of the Software Execution Function is primarily achieved by safety mechanisms integrated on the MPC5748G**.

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

- **Input / Output Functions (Application dependent)**: Input / Output functions of the MPC5748G have a high application dependency. **Functional safety will be primarily achieved by system level safety measures**.
- **Not Safety Related Functions**: It is assumed that some functions are **Not Safety Related (e.g. debug)**.

Please see the Module classification section for further details.

2.3.2 Correct operation

Correct operation of the MPC5748G is defined as:

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

2.4 Safe states

A safe state of the system is named Safe $state_{system}$, whereas a safe state of the MPC5748G is named Safe $state_{MCU}$. A Safe $state_{system}$ 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.

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

2.4.1 MCU Safe state

The safe states (Safe $state_{MCU}$) of the MPC5748G are:

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

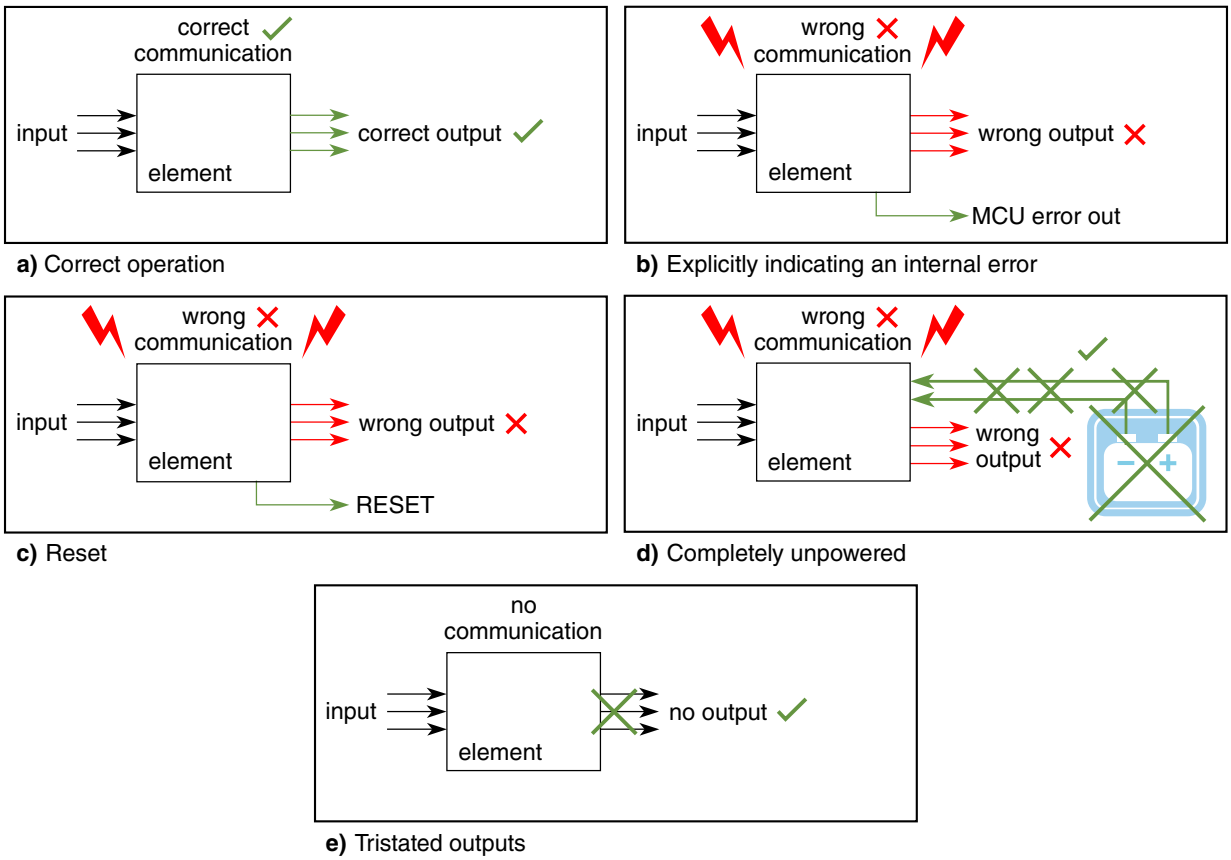

Figure 2-1. Safe state$_{MCU}$ of MPC5748G

2.4.2 Transitions to Safe state$_{system}$

Assumption: [SM_015] The system transitions itself to a Safe state$_{system}$ when the MCU explicitly indicates an internal error (as shown on FCCU_EOUT0 or FCCU_EOUT1). [end]

Implementation hint: If the MPC5748G signals an internal failure via its error out signals (FCCU_EOUT*n*), the surrounding subsystem shall no longer use the MPC5748G outputs for safety functions since these signals can no longer be considered reliable. If an error is indicated, the system shall be able to remain in a Safe state$_{system}$ without any additional action by the MPC5748G. Depending on the configuration, the system may disable, or reset, the MPC5748G as a reaction to the error signal.

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

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

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

2.4.3 Continuous reset transitions

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

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

2.5 Faults and failures

2.5.1 Failure types

Failures are the main detrimental impact to functional safety:

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

2.5.2 Faults

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

- **Single-Point Fault (SPF):** A fault in an element that is not covered by a safety mechanism, and results in a single-point failure. This leads directly to the violation of a safety goal. 'a' in the Figure 2-2 shows a SPF inside an element, which generates a wrong output. The equivalent in IEC 61508 to Single-Point Fault is a **Random fault**. Whenever a SPF is mentioned in this document, it is to be read as a random fault for IEC 61508 applications.
- **Latent Fault (LF):** A fault whose presence is not detected by a safety mechanism nor perceived by the automobile driver. A LF is a fault that does not violate the functional safety goal(s) itself, but leads to a dual-point or multiple-point failure when combined with at least one additional independent fault, which then leads directly to the violation of a functional safety goal. 'b' in the Figure 2-2 shows a LF inside an element, which still generates a correct output. No equivalent in IEC 61508 to LF is named.
- **Dual-Point Fault (DPF):** An individual fault that, in combination with another independent fault, leads to a dual-point failure. This leads directly to the violation of a functional safety goal. 'd' in the Figure 2-2 shows two LFs inside an element, which generate a wrong output.
- **Multiple-Point Fault (MPF):** An individual fault that, in combination with other independent faults, leads to a multiple-point failure. This leads directly to the violation of a functional safety goal. Unless otherwise stated, multiple-point faults are considered safe faults and are not covered in the functional safety concept of MPC5748G.
- **Residual Fault (RF):** A portion of a fault that independently leads to the violation of a functional safety goal, where that portion of the fault is not covered by a functional safety mechanism. 'c' in the Figure 2-2 shows a RF inside an element, which – although a functional safety mechanism is set in place – generates a wrong output, as this particular fault is not covered by the functional safety mechanism.
- **Safe Fault (SF):** A fault whose occurrence will not significantly increase the probability of violation of a functional safety goal. Safe faults are not covered in this document. SPFs, RFs or DPFs are not safe faults.

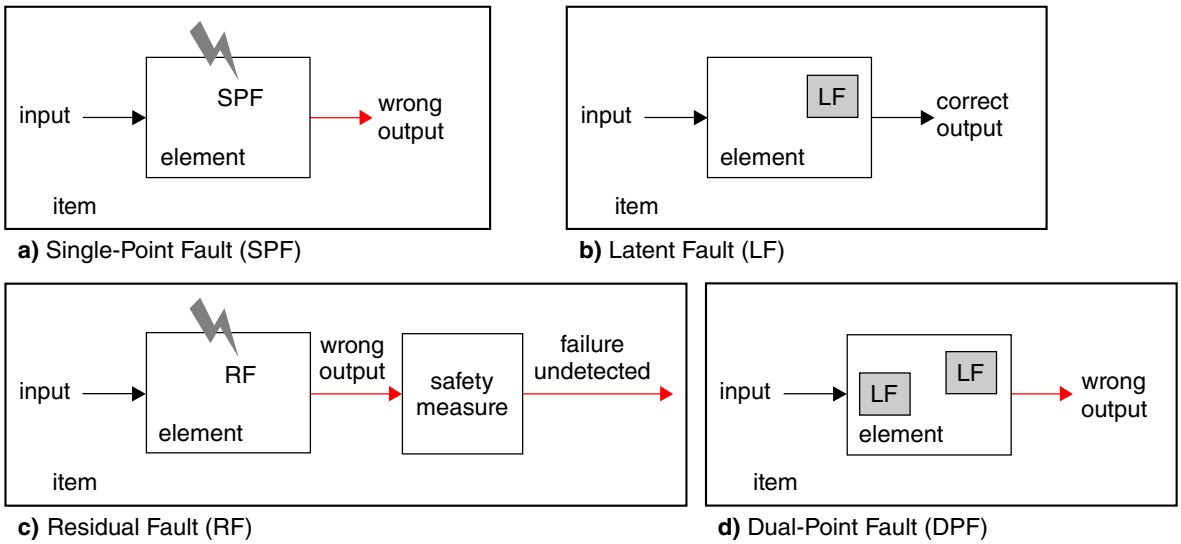

Figure 2-2. Faults

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

2.5.3 Dependent failures

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

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

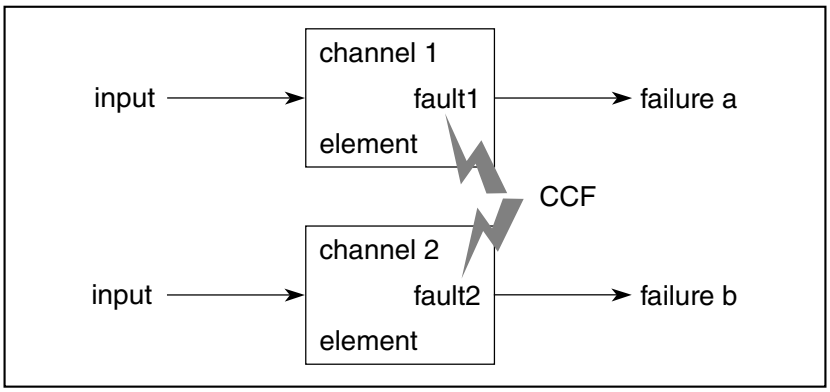

Figure 2-3. Common Cause Failures

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

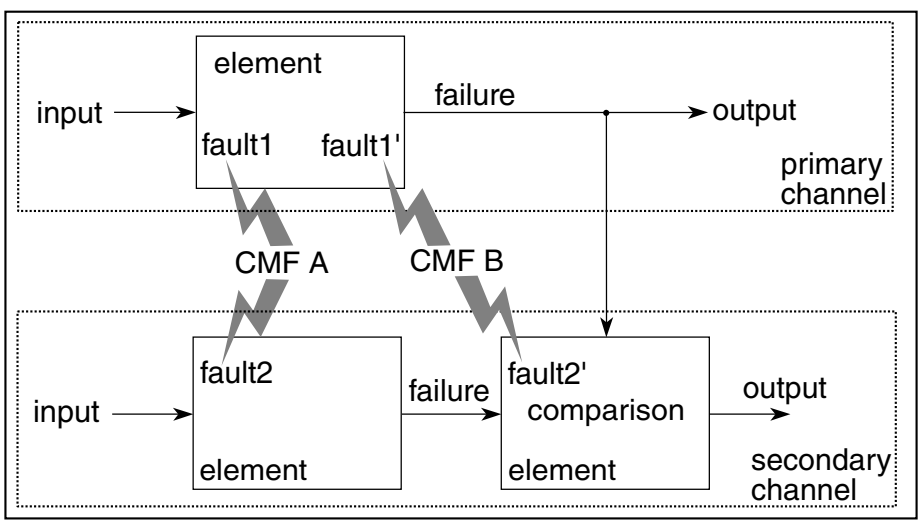

Figure 2-4. Common Mode failures

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

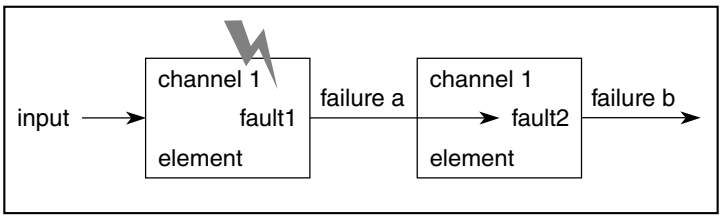

Figure 2-5. Cascading failures

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

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

Figure 2-6 shows the FTTI for a system:

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

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

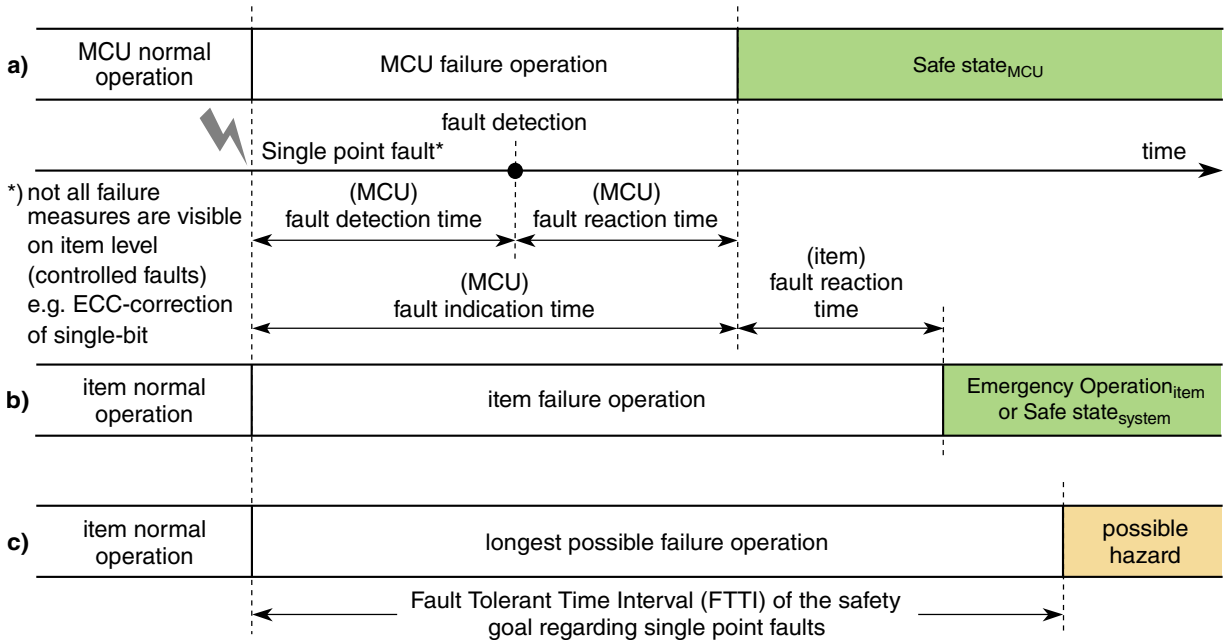

Figure 2-6. Fault tolerant time interval for single point faults

Fault indication time is the time from the occurrence of a fault to when the MPC5748G is switched into a Safe state$_{MCU}$ (for example, indication of that failure by driving the error out pins, forcing outputs of the MPC5748G to a high impedance state, or by assertion of reset).

2.6.1 MCU fault indication time

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

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

- **Internal processing time** to communicate the fault to the Fault Collection and Control Unit (FCCU), and can take up to a maximum of 10 FastInternal RC Oscillator (FIRC) clock cycles (nominal frequency of 16 MHz).
- **External indication time** to notify an observer about a failure external to the MPC5748G. This time depends on the indication protocol configured in the Fault Collection and Control Unit (FCCU):
  - Dual Rail protocol and time switching protocol:
    - **FCCU configured as "fast switching mode":** indication delay is a maximum of 64 µs. As soon as the FCCU receives a fault signal, it reports the failure to the system.
    - **FCCU configured as "slow switching mode":** an indication delay could occur. The maximum delay is equal to the duration of the semiperiod of the error out (FCCU_EOUT*n*) frequency. With an IRCOSC frequency of 16 MHz, the error out frequency is 61Hz. Therefore, the maximum indication delay 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.

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

The sum of the MPC5748G fault indication time and system fault reaction time should be less than the FTTI of the functional safety goal.

2.7 Latent-fault tolerant time interval for latent faults

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

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

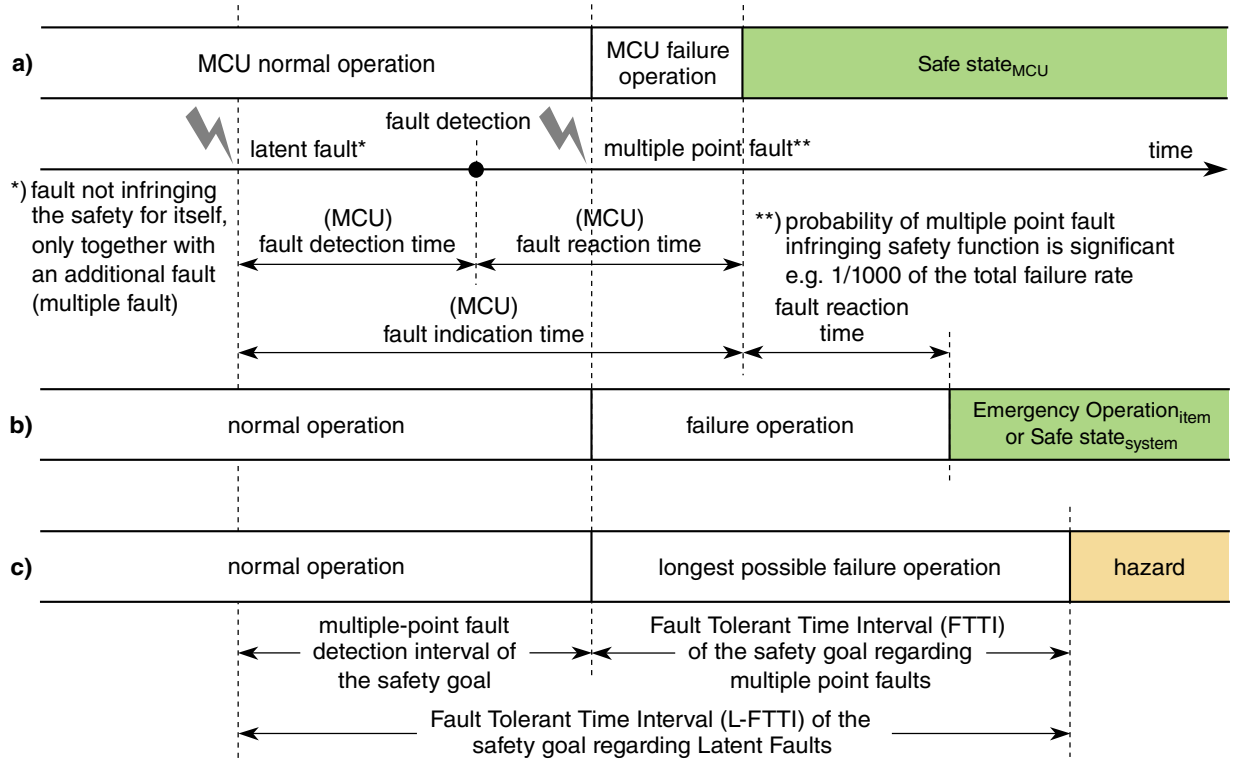

Figure 2-7. Fault Tolerant Time Interval for latent faults

Latent fault indication time is the time it takes from the occurrence of a multiple-point failure to when the indication of that failure is driven on FCCU_EOUT*n*, forcing the outputs of the MPC5748G to a high impedance state (Safe Mode) or by assertion of reset.

2.7.1 MCU fault indication time

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

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

Note

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

2.8 MCU failure indication

2.8.1 Failure handling

Failure handling can be split into two categories:

- Handling of failures before enabling the system level safety function (for example, during/following the MPC5748G initialization). These failures are required to be handled before the system enables the safety function, or in a time shorter than the respective FTTI after enabling the safety function.
- Handling of failures during runtime with repetitive supervision while the safety function is enabled. These errors are to be handled in a time shorter than the respective FTTI.

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

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

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

Software can read the failure source that caused a FCCU fault, and can do so either before or after a functional reset. Software can also reset the failure, but the external failure indication will stay in failure mode for a configurable amount of time. If necessary, software can also reset the MPC5748G.

2.8.2 Failure indication signaling

The FCCU offers a hardware channel to collect errors and bring the device to a Safe state$_{MCU}$ when a failure is present in the MPC5748G. The FCCU provides two error output signals (FCCU_EOUT0 and FCCU_EOUT1) used for external failure indication.

Different protocols for the error output pins are supported:

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

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

At least one of the FCCU_EOUT*n* outputs will be high to indicate that the device is in the operational state. If a two-pin bi-stable protocol with differential outputs is implemented (for example, FCCU_EOUT0 = 0 and FCCU_EOUT1 = 1 and vice-versa), the application software can configure that FCCU_EOUT*n* signal that will be high to indicate the operational state (see Error Out Monitor (ERRM) for details on requirements for connecting FCCU_EOUT*n* to external devices).

Chapter 3
MCU Safety Concept

3.1 General concept

Figure 3-1 is a top-level diagram showing the functional organization of the MPC5748G.

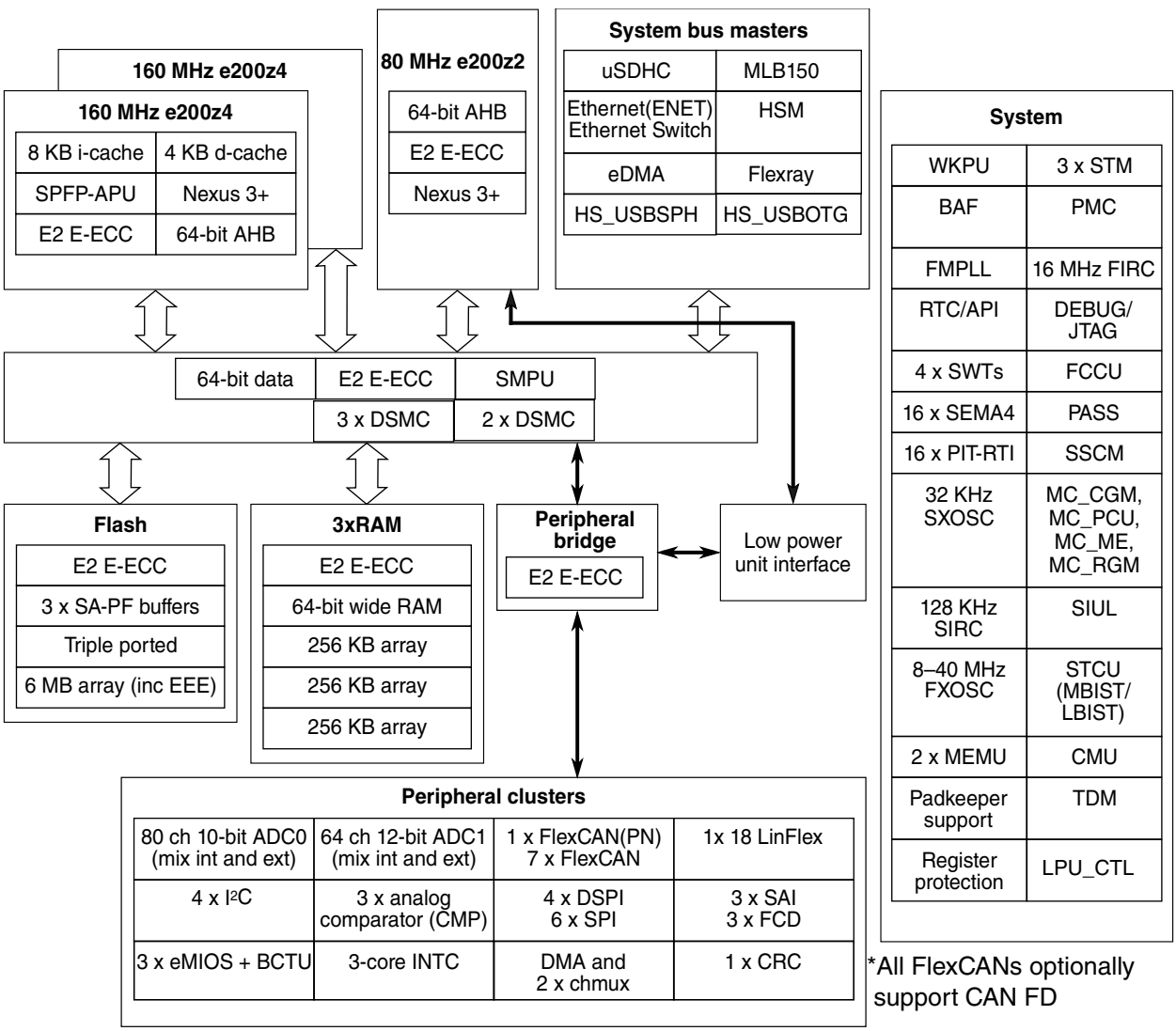

Figure 3-1. MPC5748G block diagram

The MPC5748G has an integrated safety concept targeting safety-related systems requiring high safety integrity levels. In general, safety integrity is achieved in the following ways:

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

3.2 Use of cores for safety - self-test, reciprocal comparison, temporal redundancy

In order to detect permanent faults in cores executing safety relevant code a software based self-test can be executed. The self-test can be executed independently on each of the main z4 cores. NXP has developed a Structural Core Self-Test Library for the MPC5748G e200z420 core. The Library and associated documentation is available upon request (see section Software based self-test).

To protect against permanent and transient faults, a reciprocal comparison can be executed. The application software is executed by the two core subsystems (processing units) and exchanges data (including results, intermediate results and test data) reciprocally or controls independent system channels. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message (see section Reciprocal comparison).

Another method of improving detection of transient faults is to use temporal (or time) redundancy. This is when safety-critical tasks are computed multiple times using the same or similar inputs on the same core, and the results compared within the FTTI (see section Temporal redundancy).

3.3 ECC

Error correcting codes are used for end-to-end protection from bus masters to system storage as well as for individual protection of peripheral RAMs and PBRIDGES a and b.

3.3.1 End-to-End protection on data path

Connections between AXBS masters and slaves (clients) are denoted as data paths. Data corruption on all data paths between the core and any client is detected via two main safety mechanisms: data from the cores is encoded using Error Correcting Code (ECC), which is implemented with a Single-Error Correction, Double-Error Detection (SECDED) code with a Hamming distance of 4 and includes coverage of addressing information. Control signals and address decoding are monitored to verify the data reaches all of the intended clients, from all possible connections to these clients and the intended operation is performed on the target address. Figure 3-2 illustrates the overall ECC schema.

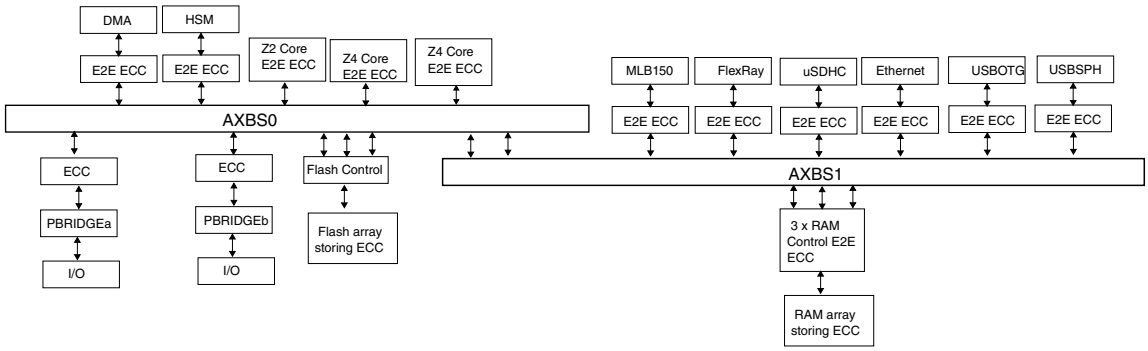

Figure 3-2. General view of e2eECC

NOTE

Specific implementations for the MPC5748G vary depending on the special requirements of RAM and flash memory concerning ECC handling, as well as for caches, local RAM, tag memories of the cores and DMA RAM.

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

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

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

Though safety mechanisms protecting the AXBS, the RAM controller, or the flash memory controller are different, they are all based on the feedback of address and control information from the target to the source of the transaction, which is responsible for checking for consistency with respect to the intended transaction. Depending on the portion of the data path covered by the specific safety mechanism, the source can be an AXBS master port rather than the AXBS interface of the RAM or Flash Memory Controllers; the target is respectively an AXBS slave port, the RAM array, or the flash memory module. See the separate *MPC5748G Reference Manual* chapters dedicated to the Crossbar Switch (AXBS), Flash Memory Controller (PFLASH), and RAM Controller (PRAMC) for further details.

NOTE

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

3.3.2 ECC for storage

All storage (System RAM, local memory, flash memory, peripheral RAM) used in normal operation is protected by ECC with SECDED (Single Error Correct and Double Error Detect). Exceptions to this are noted below:

- The caches in the Z4a and Z4b cores are protected by EDC (Error Detection Code), which detects but does not correct errors.
- FlexRay's schedule information RAM is protected by EDC only.
- The HSM's AES program extension RAM are not protected by ECC.

A list showing the implementation of RAMs with ECC (including address protection) and for the device is shown in the following table.

Table 3-1. ECC RAM Implementations

Location	Memory	Memory Classification	Address in ECC
PRAM	PRAM0	System RAM	Y
	PRAM1	System RAM	Y
	PRAM2	System RAM	Y
Z4A	z4a_icache	System RAM	Y
	z4a_itag	System RAM	Y
	z4a_dcache	System RAM	Y
	z4a_dtag	System RAM	Y
Z4B	z4b_icache	System RAM	Y
	z4b_itag	System RAM	Y
	z4b_dcache	System RAM	Y

Table continues on the next page...

Table 3-1. ECC RAM Implementations (continued)

Location	Memory	Memory Classification	Address in ECC
	z4b_dtag	System RAM	Y
HSM	hsm_PRAM	Peripheral RAM	Y
	hsm ic_data	Peripheral RAM	Y
	hsm ic_tag	Peripheral RAM	Y
FlexRay	flxray_data	Peripheral RAM	Y
	flxray_lut	Peripheral RAM	Y
DMA	dma_ram	Peripheral RAM	Y

Some memories, particularly system storage, use an ECC computed over data and address to detect data and addressing faults (no/wrong/multiple selection). In addition, these same memories include dedicated measures against addressing and control faults (such as address/control feedback). This is different for storage related to peripheral modules. Peripheral modules in general use an ECC without address error protections and also do not include additional measures to detect them.

3.3.3 All-X words and ECC

There is a special case for legal ECC values in the MPC5748G. Memory entries that are all zeros (All-0) or all ones (All-1), including the ECC parity bits, are not legal for memory that is checked by ECC. The flash memory allows All-1, corresponding to the status of an erase block, as a valid codeword.

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

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

Notice that for flash memory, additional dedicated safety mechanisms exist to detect failures that have the potential of leading to an All-1 word (see the "Flash Memory Controller (PFLASH)" chapter in the *MPC5748G Reference Manual* for more details on flash memory safety mechanisms).

3.3.4 ECC failure handling

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

3.4 Clock and power monitoring

3.4.1 Clock

Clocks in the MPC5748G are supervised by a Clock Monitor Unit (CMU). The CMU is driven by the FIRC (16 MHz internal oscillator) for independent operation from the monitored clocks. If a supervised clock exceeds or falls below its specified frequency range on the chip, the supervising CMU flags an error that sends a signal to the FCCU.

Clocks supervised by CMU instance are as follows:

- CMU: FIRC, FMPLL, and FXOSC

NOTE

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

3.4.2 Power

There are two types of voltage supervisors on the MPC5748G: Low Voltage Detect (LVD) and High Voltage Detect (HVD) monitors. Safety relevant voltages (recommended operating voltages) are supervised for values that are out of these ranges. Since any voltage running outside of the safety relevant range has the potential to disable the failure indication mechanisms of the MCU (such as FCCU, pads, and so on), the indication of these errors can be used to cause a direct transition of the MCU into the safe state (reset assertion) (see the "Power Management Controller block (PMC)" and "Power Control Unit (MC_PCU)" chapters in the *MPC5748G Reference Manual* for details).

3.5 I/O peripherals

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

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

3.6 Communication controllers

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

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

- FlexRay
- FlexCAN
- Ethernet
- MediaLB
- uSDHC
- USBOTG

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

3.6.1 Disabling of communication controllers

In the event of a dangerous failure, the MCU offers the capability of disabling transmission of individual channels of communication controllers such as:

- CAN
- FlexRay

Such disabling prevents the transmission of erroneous messages while preserving the capability of communicating over the diagnostic bus. Disabling outputs is controlled by resetting SIUL2_MSCR*n*[SMC] for the pins that are associated with communication controllers where this feature is needed (see the "Pin muxing" table and the SIUL2 Multiplexed Signal Configuration register description in the "System Integration Unit Lite2 (SIUL2)" chapter for details, as shown in the *MPC5748G Reference Manual*).

The FCCU intends to drive FCCU_EOUT0 to a fault state whenever FCCU FSM is in fault state or FCCU_CFG[FCCU_SET_CLEAR] is 01b. When the FCCU intends to drive FCCU_EOUT0 to a fault condition, the SIUL2 disables the output buffer of such pins for which SIUL2_MSCR*n*[SMC] is cleared and thus disables transmission of erroneous messages until FCCU intends to drive FCCU_EOUT0 to a non-fault condition. After a communication controller transmission port is disabled, it remains in the same state as long as the FCCU drives FCCU_EOUT0 to a non-fault condition. During this mode, the state of weak pull-up/pull-down remain unchanged.

The application should configure SIUL2_MSCR*n*[SMC] for pins that have active mapping of communication module (for example, FlexCAN, FlexRay) functionality and ensure those pins do not remain in an undriven state.

NOTE

The FCCU uses internal signals to disable the communication controller transmission, so that transmission is disabled even when the FCCU cannot drive the EOUT0 pin because the pin is configured as GPIO. Also, if EIN is used as error input, externally pulling it up will disable the communication controller transmission only if this event drives FCCU to fault state.

3.7 Built-In Self Tests (BIST)

The term BIST indicates the set of built-in hardware mechanisms that can be used (typically at startup) to avoid the accumulation of latent faults. BIST is a mechanism that permits a device to test itself. On the MPC5748G, BIST is the main means to meet the requirement on latent faults as defined by the ISO 26262 standard. Different types of

BIST are implemented in the MPC5748G: LBIST for digital logic, MBIST for memories, and the MCU's built-in mechanisms for testing analog peripherals. LBIST and MBIST execution is managed by the STCU2, while the testing of analog peripherals requires software intervention to be triggered (see chapter "Self-Test Control Unit (STCU2)" in the *MPC5748G Reference Manual*).

3.7.1 BIST during boot

It is possible to configure the device such that a device BIST is performed every time the device boots. BIST is performed transparently for the application while the device is still under reset. In case the BIST fails, it is possible to configure the device to remain in reset (refer to the SUF_DIS bit in the UTEST Miscellaneous register). Application software can start executing when the BIST finishes successfully without detecting a fault. The boot time BIST comprises:

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

A destructive reset should be triggered at least once per L-FTTI (for example, once per drive cycle) to ensure an offline LBIST is performed. In some applications, the MPC5748G may not be reset within the L-FTTI but may instead enter Low Power mode within the L-FTTI. In this case, a destructive reset can be triggered upon exit from Low Power mode, triggering an offline LBIST.

3.7.2 LBISTed modules

NOTE

At the end of LBIST routine execution, the modules present inside LBIST partitions will completely get reset (i.e the behavior will be similar to a power on reset).

LBIST is implemented for safety-critical modules. There are three LBIST controllers in the device for following partitions:

Table 3-2. LBISTed modules

Partition 2 - Z4	Partition 0 - LPU	1 - Non-core bus masters and crossbar within platform
z4a core	MEMU_1	MEMU_0

Table continues on the next page...

Table 3-2. LBISTed modules (continued)

Partition 2 - Z4	Partition 0 - LPU	1 - Non-core bus masters and crossbar within platform
z4b core	PCM	HSM
	z2 core	MLB150
	PBRIDGE_A	uSDHC
	-	USB_0 and USB_1
	STM_0	FlexRay
	PRAM_0	SMPU_0
	INTC	SMPU_1
	ADC_0	Z2 iAHB gasket
	EMIOS_2	PBRIDGE_A iAHB gasket
	DSPI_4	PBRIDGE_B
	LINFlex_0	PBRIDGE_B iAHB gasket
	FlexCAN_0	-
		eDMA iAHB Gasket
		ENET iAHB Gasket
		ENET e2e ECC Gasket
		Flash memory controller
		FlexRay e2e ECC Gasket
		FlexRay iAHB Gasket
		HSM iAHB Gasket
		HSM e2e ECC Gasket
		MLB iAHB Gasket
		MLB e2e ECC Gasket
		PRAM_1
		PRAM_2
		uSDHC iAHB Gasket
		uSDHC e2e ECC Gasket
		Semaphores2
		STM_1
		STM_2
		SWT_1
		SWT_2
		USB_0 iAHB Gasket
		USB_0 e2e ECC Gasket
		USB_1 iAHB Gasket
		USB_1 e2e ECC Gasket
		eDMA
		AXBS_1
		AXBS_0
		AXBS_0 iAHB Gasket

Table continues on the next page...

Table 3-2. LBISTed modules (continued)

Partition 2 - Z4	Partition 0 - LPU	1 - Non-core bus masters and crossbar within platform
		AXBS_1 iAHB Gasket
		DMAMUX_0
		DMAMUX_1
		TRNG_0
		TRNG_1
		FCCU

3.8 FCCU and failure monitoring

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

3.8.1 External error indication

Failure of the MCU is signaled to one or two pins, FCCU_EOUT0 and FCCU_EOUT1. FCCU_EOUT0 can also serve as an error input mechanism (see Fault Collection and Control Unit (FCCU) and the FCCU configuration section in the *MPC5748G Reference Manual* for details on the fault output signals).

The error indication on pins FCCU_EOUT0 and FCCU_EOUT1 is controlled by the FCCU.

NOTE

FCCU EOUT0 and EOUT1 are muxed on the pad, and their controls will be via SIUL. There will be no interaction with the HSM for this control.

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

EOUT does not indicate a fault condition if the chip is in reset. When the chip is in reset and the I/O is high-impedance, you must assure the system safe state; for example, using pull-up/down resistors to pull EOUT to its fault indication level.

3.8.2 Failure handling

The FCCU is an autonomous module that is responsible for reacting to failure indicators. A different reaction can be configured for each failure source. Overall failure reaction time requires time for detecting, processing, and indicating the error. During this time, the MPC5748G could provide incorrect results to the system.

Failure sources include:

- All failure indication signals from modules within the MCU
- Control logic and signals monitored by the FCCU itself.
- Software-initiated failure indications. For example, software signals the FCCU that it has evidence of a failure. Keep in mind that software can also directly influence the state of the FCCU_EOUT*n* pins.
- External failure input

Available failure reactions are:

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

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

3.8.3 Fault inputs

The table "FCCU Non-Critical Faults Mapping" in chapter "Fault Collection and Control Unit (FCCU)" of the *MPC5748G Reference Manual* shows the source of the fault signals and the type of fault input to which these signals are connected at the FCCU.

3.9 Memory Error Management Unit (MEMU)

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

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

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

As the MEMU aggregates address information from several sources, it might not be able to process all simultaneously arriving reports. This is called a simultaneous overflow and is signaled to the FCCU as a buffer overflow (see the "Memory Error Management Unit (MEMU)" chapter in the *MPC5748G Reference Manual*).

3.9.1 Interface to ECC units

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

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

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

NOTE

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

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

NOTE

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

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

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

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

3.10 Operational interference protection

Being a multi-master system, MPC5748G provides safety mechanisms to prevent non-safety masters from interfering with the operation of the core, as well as mechanisms to handle the concurrent execution of software with different (lower) ASIL. Interference freedom is guaranteed via a hierarchical memory protection schema including:

- SMPU
- PBRIDGEs
- Register protection.

Memory protection is provided by the System MPUs (SMPU) located in each AXBS. They will prevent access of different bus masters to address ranges and will typically be used by the safety application to prevent non-safety related modules access to the application's safety-relevant resources.

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

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

These safety mechanisms are further described in the SMPU chapter of the MPC5748G Reference Manual.

3.11 Common cause failure measures

Various measures are included to prevent common cause failures (CCF) from endangering the effectiveness of the core or of the peripherals. These measures include supervision of clock frequency and voltage level signals. In general, these measures are independent from the software.

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

Chapter 4
Hardware Requirements

4.1 Hardware requirements on system level

This section describes the system level hardware safety measures needed to complement the integrated safety mechanisms of the MPC5748G.

The MPC5748G integrated safety concept enables SPFs and latent failures to be detected with high diagnostic coverage. However, not all CMFs may be detected. In order to detect failures which may not be detected by the MPC5748G, it is assumed that there will be some separate means to bring the system into Safe $state_{system}$.

Figure 4-1 depicts a simplified application schematic for a functional safety-relevant application in conjunction with an external IC (only functional safety related elements shown). The supplies generated from the external IC should be protected against voltage over the absolute maximum rating of the device (as documented in the MPC5748G Data Sheet in section "Absolute maximum ratings").

The external circuit will also monitor the FCCU_EOUT*n* signals. Through a digital interface (for example, SPI), the MPC5748G repetitively triggers the watchdog of the external IC. If there is a recognized failure (for example, watchdog not being serviced, assertion of FCCU_EOUT*n*), the reset output of the external IC will be asserted to reset the MPC5748G. A fail-safe output is also available to control or deactivate any fail-safe circuitry (for example, power switch).

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

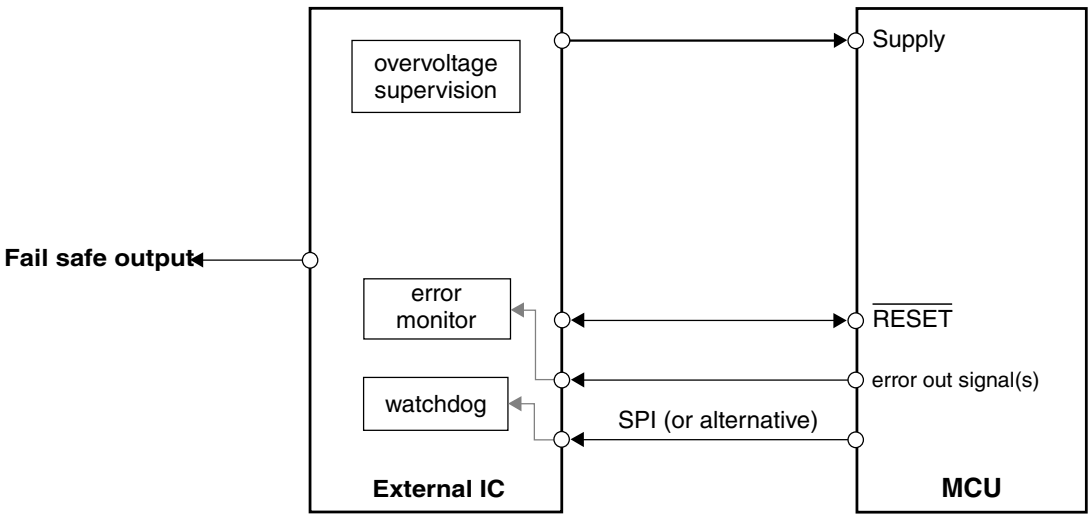

Figure 4-1. Functional safety related connection to external circuitry

4.1.1 Assumed functions by separate circuitry

This section describes external components used in a system in conjunction with the MPC5748G for safety-related systems.

It should be noted that failure modes of external services are only partially considered in the FMEDA of the MPC5748G (for example, clock(s), power supply), and must be fully analyzed in the system FMEDA by the safety system developer.

4.1.1.1 High impedance outputs

If the MPC5748G is considered to be in a Safe $state_{MCU}$ (for example, unpowered and outputs tristated), the system containing the MPC5748G may not be compliant with the Safe $state_{system}$. A possible system level safety measure to achieve Safe $state_{system}$ may be to place pull-up or pull-down resistors on I/O when the high-impedance state is not considered safe.

Assumption: [SM_038] If a high-impedance state on an output pin is not safe, pull-up or pull-down resistors shall be added to safety-related outputs. The need for this will be application dependent for the unpowered or reset (tristated I/O) MPC5748G.[end]

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

4.1.1.2 External Watchdog (EXWD)

An external device, acting as an independent timeout functionality (for example, External Watchdog (EXWD)), should be used to cover Common Mode Failures (CMF) of the MPC5748G for safety-related systems.

The trigger may be a discrete signal(s) or message object(s). If within a defined timeout period the EXWD is not triggered, a failure will be considered to have occurred which would then switch the system to a Safe $state_{system}$ within the FTTI (for example, the EXWD disconnects the MPC5748G 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 MPC5748G and the EXWD can be chosen by the user as warranted by the application. Examples of different mechanisms that can be used to trigger the EXWD can include any of the following:

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

4.1.1.3 Power Supply Monitor (PSM)

Supply voltages outside of the specified operational ranges may cause permanent damage to the MPC5748G, even if it is held in reset.

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

The *MPC5748G Microcontroller Data Sheet* provides specific operating voltage ranges that must be maintained.

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

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

If the power supply is out of range, *MPC5748G* shall be kept in reset or unpowered, or other measures must possibly be used to keep the system in a safe state. Overvoltage outside the specified range of the technology may cause permanent damage to the *MPC5748G* even if kept in reset.

Implementation hint: An external and independent device may provide an over voltage monitor for the external MPC5748G supplies. If the supplied voltage supply is above the recommended operating voltage range of the MPC5748G, the MPC5748G 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 MPC5748G power supply can be designed to avoid any potential of over-voltage, the external voltage monitoring can be excluded from the system design.

Over-voltage on some supplies will be detected by the MPC5748G itself, but system level measures might be required to maintain the Safe $state_{system}$ in case an over-voltage situation may cause damage to the MPC5748G.

4.1.1.4 Error Out Monitor (ERRM)

If the MPC5748G signals an internal failure on its error out signals (FCCU_EOUT0, and/or FCCU_EOUT1), the system may no longer rely on the integrity of the other MPC5748G 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 MPC5748G. Depending on its functionality, the system might disable or reset the device as a reaction to the error indication (see **Assumptions** in Safe states).

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

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

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

4.1.1.4.1 Both FCCU signals connected to separate device

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

This configuration does not require any dedicated software support.

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

NOTE

See “EOUT interface” section in the “Fault Collection and Control Unit (FCCU)” chapter of the *MPC5748G Reference Manual* for details.

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

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

4.1.1.4.2 Single FCCU signal connected to separate device

A single signal, FCCU_EOUT0 (or FCCU_EOUT1), is connected to a separate device.

If a fault occurs, the FCCU communicates the fault to the separate device through the FCCU_EOUT0 (or FCCU_EOUT1).

The functionality of FCCU_EOUT0 (or FCCU_EOUT1) can be checked in the following manner:

- FCCU_EOUT0 (or FCCU_EOUT1) read back internally.
- FCCU_EOUT0 (or FCCU_EOUT1) connected externally to a GPIO.
- FCCU_EOUT0 (or FCCU_EOUT1) uses time domain coding (for example, is active for a deterministic time interval).
- Test the ability of FCCU_EOUT0 (or FCCU_EOUT1) to disable system functionality (for example, measure voltage available at a motor if FCCU_EOUT0 (or FCCU_EOUT1) is expected to disable its power supply).

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

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

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

Recommendation: If FCCU_EOUT0, or FCCU_EOUT1, is connected to a device not using time domain coding, verification is needed that the FCCU_EOUT*n* signal(s) are operating correctly before execution of any safety function can start.

Rationale: To check the integrity of FCCU_EOUT0, or FCCU_EOUT1

To verify the functionality of a FCCU_EOUT*n* signal, a fault may be injected into one of the FCCU_EOUT*n* signals. The behavior of the signal can then be verified by the other FCCU_EOUT*n* signal, or GPIO. Additionally, the fault output mode can be configured to one of the test modes to control one FCCU_EOUT*n* as an output while the other FCCU_EOUT*n* pin is an input or output. For example, TEST0 mode configures FCCU_EOUT0 as an input and FCCU_EOUT1 as an output. This test mode can be used to check the state of the FCCU_EOUT0 input by reading FCCU_EINOUT[EIN0]. Likewise, the user can control the FCCU_EOUT1 output by modifying FCCU_EINOUT[EOUT1].

Since the FCCU will be monitoring the system, it is sufficient to check FCCU_EOUT0 (or FCCU_EOUT1) within the L-FTTI (for example, at power-up) to help reduce the risk of latent faults. It is recommended that FCCU_EOUT*n* be checked once before the system begins performing any safety-relevant function.

Assumption: [SM_170a] If the system is using the MCU in a single error output configuration, the application software will need to configure the signals, and pads, adjacent to FCCU_EOUT0 (or FCCU_EOUT1) to have a lower drive strength, and the error output signal is configured with highest drive strength. [end]

Using a lower drive strength on the GPIO near FCCU_EOUT0 (or FCCU_EOUT1) will result in the higher drive strength of FCCU_EOUT*n* to effect the logic level of the neighboring GPIO in the event of a short circuit. Software may configure the slew rate for the relevant GPIO in the Multiplexed Signal Configuration Register (SIUL2_MSCR*n*) and Input Multiplexed Signal Configuration Register (SIUL2_IMCR*n*).

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

Rationale: Decode the time domain coding

Implementation hint: If a single FCCU signal (FCCU_EOUT0, or FCCU_EOUT1), is connected to a separate device applying time domain coding (for example, a decoder), a window timeout or windowed watchdog function, is good practice.

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

4.1.2 Optional hardware measures on system level

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

4.1.2.1 External communication

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

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

4.1.2.2 PWM output monitor

The MPC5748G timer modules may require system-level safety measures in order to achieve high functional safety integrity levels.

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

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

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

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

- Dead-time may need to always be positive, and greater than the maximum value of T_{ON} or T_{OFF} of the inverter switches.
- Open GPIO, and shorts to supply or ground, may need to be detected. This can be accomplished, for example, by an external feedback mechanism to a timer module of the MPC5748G capable of performing input capture functionality.

The system must be switched to Safe $state_{system}$ if the MPC5748G detects an error.

To reduce the likelihood of erroneous control (for example, a motor control application with dead-time requirements to reduce the likelihood of short circuits destroying the motor) in functional safety applications using I/O to control an actuator with a short FTTI, functional safety requires system level supervision if the maximum fault indication time and fault reaction time of MPC5748G exceeds the FTTI of the actuators.

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

Chapter 5
Software Requirements

5.1 Software requirements on system level

This section lists required, or recommended, measures when using the individual components of MPC5748G.

Given the application independent nature of the MPC5748G, no general safety function can be specified. To define a specific safety function the MPC5748G would have to be integrated into a complete (application dependent) system. Nevertheless, it is possible to define abstract safety function elements and safety integrity functions:

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

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

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 MPC5748G 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 MPC5748G may not enter debug mode.

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

Assumption under certain conditions:[SM_048] If modules like the Software Watchdog Timer (SWT), System Timer Module (STM), Deserial Serial Peripheral Interface (DSPI), Periodic Interrupt Timer (PIT), Fault Collection Control Unit (FCCU), FlexRay, 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

5.1.1.2 Test mode

Several mechanisms of the MPC5748G can be circumvented during test mode which endangers the functional safety integrity.

Assumption: [SM_149] Test mode is used for comprehensive factory testing and is not valid for normal operation. Test mode may not be used during normal operating mode without an explicit agreement from NXP Semiconductors. [end]

Implementation hint: The VSS_HV_VPP 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 VSS_HV_VPP pin is not connected to V_{DD} during boot to avoid entering test mode. The activation of test mode is supervised by the FCCU and will signal a fault condition when test mode is entered. FIRC clock-related test mode activation is intended to be covered by the frequency meter function of CMU_0, as described in the FIRC Runtime checks section.

5.2 MPC5748G modules

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

Assumption:[SM_151] It is assumed that communication over FlexRay and FlexCAN interfaces be protected by a fault-tolerant communication protocol.[end]

The high-bandwidth communication controllers will not contain special safety mechanisms (above what is included into them by their protocol specifications) nor are they required to be duplicated or spread over the PBRIDGEs' (in contrast to other I/O, see Redundant communication).

The high-bandwidth communication controllers effected by the above requirement shall be listed to the customer.

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

Recommendation: Some redundant software checks (for example, a timer reset) after periodic messages are received ensures communications remain active for critical channels. This may be acknowledged by a core other than the main core. This helps to avoid delays in the main core.

5.2.1 Cores

The cores Main Core 0 and Main Core 1 (each an e200z420) may be used to process non-safety critical tasks as well as safety-critical tasks. When executing safety critical tasks, it may be required to add software-based safety measures in order to achieve the desired level of safety integrity. For each z4 core, faults in the ALU data path, registers (e.g. general purpose registers), the address calculation (e.g. in the load/store unit or memory and bus interfaces), the bus interface or the control logic (e.g. the sequencer, coding and execution logic) could potentially affect correct execution of code, so software safety measures should be put in place to detect these faults.

5.2.1.1 Runtime checks

5.2.1.1.1 Reciprocal comparison

To protect against permanent and transient faults, a reciprocal comparison should be executed. The application software is executed by the two core subsystems (processing units) and exchanges data (including results, intermediate results and test data) reciprocally or controls independent system channels. A comparison of the data is carried out using software in each unit and detected differences lead to a failure message.

Assumption:[SM_305] It is assumed that application software reciprocal comparison shall be used to check the safety integrity of the cores executing safety-critical tasks. [end]

5.2.1.1.2 Software based self-test

A second safety measure to detect permanent faults is to execute a software based self-test. The self-test can be executed independently on each of the z4 cores. Within the FTTI, the processing unit shall periodically run code that tests the functionality of the processing unit (structural test). The result of this software based self-test is calculated offline during development and compared with the real-time value.

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

Preferably, the processing unit should not be the only method used to compare these two values (self-test). Additionally, independent hardware should check that the result is correct.

Implementation hint: Independent hardware could be:

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

Implementation hint: NXP has developed a Structural Core Self-Test Library for the MPC5748G e200z420 core. The Library and associated documentation is available upon request.

5.2.1.1.3 Temporal redundancy

One method of improving detection of transient faults is to use temporal (or time) redundancy. This is when the safety function is computed multiple times using the same or similar inputs on the same core, and the redundant results compared within the Fault Tolerant Time Interval. Any mismatch or result comparison out of an acceptable range of the execution results is flagged as an error condition. This method has very low coverage of permanent faults, or transient faults that have a duration longer than the FTTI. However, it has good coverage of single point faults. In order to eliminate the common cause failure of executing corrupted code in cache, it is necessary to flush the caches between executions.

It is the responsibility of application software to perform a comparison of the same software running repeatedly on the same core, and flag an error when necessary.

5.2.2 Fault Collection and Control Unit (FCCU)

The FCCU uses a hardware fail safe interface which collects faults and brings the device to a Safe $state_{MCU}$, and indicates the internal failure of the MCU to the system, when a failure is recognized.

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

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

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

The table “FCCU non-critical fault mapping” in the *MPC5748G Reference Manual* shows the sources for critical faults to be signaled to the FCCU and the type of issued reset.

The failure of the MCU shall be signaled using two pins. These pins can be muxed with other signals.

The FCCU has two multiplexed external signals, FCCU_EOUT0 and FCCU_EOUT1, through which critical failures are reported. When the device is in reset or unpowered, these outputs are tristated. The FCCU will provide notification via multiplexed pins an error condition. This is to notify external support circuitry that an error occurred and respective tasks can be taken.

The failure indication pin(s) shall be push-pull pins.

If the MCU is used in a single error out pin mode, the FCCU_EOUT0 pin will be used.

The pads of the error out pins shall have the highest drive strength available on the MCU (other pins are also allowed to have this drive strength).

The pads adjacent to the FCCU_EOUT0 pad, as well as the pad of the pins adjacent to the FCCU_EOUT0 pin, (in case that differs from the pad arrangement) shall be able to be configured for a lower drive strength than the highest possible.

No VDD pin/pad shall neighbour the FCCU_EOUT0 pin/pad.

FCCU_EOUT*n* are intended to be connected to an independent device which continuously monitors the signal(s). If a failure is detected, the separate device switches to and maintains the system to a Safe $state_{system}$ condition within the FTTI (for example, the separate device disconnects the MPC5748G device or an actuator from the power supply).

5.2.2.1 Initial checks and configurations

Besides the possible initial configuration, no intervention from the MPC5748G is necessary for fault collection and reaction.

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

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

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

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

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

Assumption under certain conditions:[SM_154] If the outputs of the system I/O need to be forced to a high impedance state upon entering safe mode, MC_ME_SAFE_MC[PDO] = 1 needs to be written. [end]

Assumption:[SM_272] If the MPC5748G signals an internal failure via its error out signals (FCCU_EOUT*n*), the system can no longer safely use the MPC5748G 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 MPC5748G. Depending on its functionality, the system might disable or reset the MPC5748G as a reaction to the indicated error. [end]

5.2.2.2 Runtime checks

Assumption under certain conditions:[SM_155] If the MPC5748G is continuously switching between a standard operating state and reset, or fault state, without a device 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 device from a fault state to normal state. This procedure may help avoid the possible looping between functional and fault states.

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

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_248] Before resetting the reset counters, the application software shall ensure that it can detect longer reset cycles caused by faults in normal operation. [end]

NOTE

Longer reset cycles means length of time since the previous reset.

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

5.2.3 Reset Generation Module (MC_RGM)

5.2.3.1 Initial checks and configurations

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

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

To trigger a reset of the device by software, the MC_ME_MCTL[TARGET_MODE] shall be used. Writing MC_ME_MCTL[TARGET_MODE] = 0000b causes a functional reset where writing MC_ME_MCTL[TARGET_MODE] = 1111b causes destructive reset (see section "Reset Generation Module (MC_RGM)" of the *MPC5748G Reference Manual* for details).

5.2.3.1.1 Consecutive resets

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

To detect a loop of resets, the MPC5748G supports functional reset escalation which can be used to generate a destructive reset if the number of functional resets reaches the programmed value. Once the functional reset escalation is enabled, the Reset Generation Module (MC_RGM) increments a counter for each functional reset that occurs between writes to the MC_RGM_FRET register. When the number of functional resets reaches the programmed value in the MC_RGM_FRET, the MC_RGM initiates a destructive reset. The counter can be cleared by software, destructive reset or power-on reset.

Assumption: [SM_279] Safety software shall reset the destructive reset counter everytime after a reset, and is certain the MCU is working correctly. [end]

Assumption: [SM_159] The application software should reset the functional reset counter every time it has finished checking its environment during startup. [end]

Assumption: [SM_160] Software must ensure that the counter threshold in the MC_RGM_FRET and MC_RGM_DRET registers is a non-zero value. The default setting of both MC_RGM_FRET and MC_RGM_DRET is 0xFh. [end]

5.2.4 Self Test Control Unit (STCU2)

The STCU2 executes built-in self-test (LBIST, MBIST) and gives reaction to detected faults by signaling faults to either the MC_RGM or to the FCCU (see "Self-Test Control Unit (STCU2)" in the *MPC5748G Reference Manual* for details).

5.2.4.1 Initial checks and configurations

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

Assumption under certain conditions:[SM_162] When built in self test (for example, LBIST, MBIST) circuits of the MPC5748G are used as functional safety integrity measure (for example, to detect random faults, latent fault detection, and single-point fault detection) in a functional safety system, functional safety integrity measures on system level shall be implemented ensuring STCU2 integrity during/after STCU2 initialization but before executing a safety function. [end]

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

Implementation hint: System (application) level software shall carry out checking of STCU2 for ensuring STCU2 integrity (see section "Integrity software operations" in "Self-Test Control Unit (STCU2)" chapter in the *MPC5748G 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 increasing the STCU2 self-test coverage.

Implementation hint: The STCU 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 *MPC5748G Reference Manual*).

For power-on-reset, LVD selftest software has to be able to handle resets caused by the power-on-reset selftest.

If one of the selftest fails, either hardware (preferred) or software shall prevent normal operation to start. Failure in this context means:

- Any error in the LBIST.
- MBIST reports any multi-bit error or more than X single-bit errors.
- Any error in any other selftest module (in case they exist) including the selftest controller itself.

If the MEMU is used to count MBIST error reports, X shall be a number smaller than the size of the MEMU instance used to count MBIST error reports.

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 excessively long or short period of time. Once the SWT is enabled, it requires periodic and timely execution of the watchdog servicing procedure. The service procedure must be performed within the configured time window, before the service timeout expires. When a timeout occurs, a trigger to the FCCU can be generated immediately, or the SWT can first generate an interrupt and load the down-counter with the timeout period. If the service sequence is not written before the second consecutive timeout, the SWT drives its FCCU channel to trigger a fault (see FCCU mapping of faults).

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

Assumption:[SM_302] The SWT time window settings must be set to a value less than the FTTI/PST. Detection latency shall be smaller than the FTTI/PST.[end]

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

Implementation hint: To enable the SWT and to hard-lock the configuration 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/PST.

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

The MPC5748G 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 highly effective temporal flow monitoring.

The assumptions of the STM module of crosscheck with the PIT and the SWT must be followed.

Assumption: [SM_169] It is the responsibility of the application software to insert control flow checkpoints with the required granularity as required by the application. [end]

A valid service procedure re-loads the down counter with the time-out period.

Two service procedures are available:

- A fixed service sequence represented by a write of two fixed values (A602h, B480h) to the SWT service register. Writing the service sequence reloads the internal down counter with the timeout period.
- The next procedure is based on a pseudo-random key computed by the SWT every time it is serviced and is written by 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 $SK_{(n+1)} = (17 \times SK_n + 3) \bmod 2^{16}$.
- Fixed address execution – Watchdog is serviced by executing code at the address loaded into the designated IAC register, which cannot be updated while the watchdog is enabled.
- Incremental address execution – Watchdog is serviced by executing code at the address loaded into the designated IAC register, which can be updated.

The SWT down counter is always driven by the SIRC 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 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).

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.

5.2.6.1 Runtime checks

Portions of the MPC5748G configuration registers do not provide the functional safety integrity IEC 61508 series and ISO 26262 requires for high functional safety integrity targets on their own. This relates to systematic faults (for example, application software incorrectly overwriting registers), as well as random hardware faults (bit flipping in registers).

Assumption: [SM_170b] The CRC calculation shall be executed at least once per FTTI to verify the content of the safety-relevant configuration registers. [end]

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

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

Note

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

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 MPC5748G configuration registers of the modules involved with the safety function, the CRC module may be used to calculate a signature of the content of the registers and compare this signature with a value calculated during development.

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

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

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

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

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

5.2.6.1.1 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.

5.2.6.1.1.1 <*module*>_SWTEST_REGCRC

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

- EMIOS0_SWTEST_REGCRC

  The eMIOS0 configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- EMIOS1_SWTEST_REGCRC

  The eMIOS1 configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- EMIOS2_SWTEST_REGCRC

  The eMIOS2 configuration registers are read and a CRC checksum is computed. The checksum is compared with the expected value.

- SIUL_SWTEST_REGCRC

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

- 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.

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

5.2.7 Slow Internal RC Oscillator

The Slow Internal RC Oscillator (SIRC) has a nominal frequency of 128 kHz, but the frequency accuracy over the entire voltage and temperature ranges must be considered (see the *MPC5748G Data Sheet* for temperature and voltage variation details).

Functional safety-related modules that use the SIRC clock are:
- SWT*n*

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

5.2.8 Fast Internal RC Oscillator (FIRC)

The Fast Internal RC Oscillator (FIRC) has a nominal frequency of 16 MHz, but the frequency accuracy over the full voltage and temperature ranges must be taken into account (see the *MPC5748G Data Sheet* for temperature and voltage variation details).

Functional safety-related modules that use the FIRC clock are:
- FCCU
- CMU

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

As stated in PLL Digital Interface (PLLDIG), the FIRC should not be used as the input of the PLL for the system clock of the MPC5748G.

5.2.8.1 Initial checks and configurations

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

Assumption: [SM_173] The FIRC frequency is measured and compared to the expected frequency of 16 MHz. This test is performed after power-on, but before executing any safety function. Software writes CMU_CSR[SFM] = 1 to start the frequency measurement, and the status of the measurement is checked by reading this same field. When as CMU_CSR[SFM] = 0 the frequency measurement has completed (see "Frequency meter" section in the "Clock Monitor Unit (CMU)" chapter of the *MPC5748G Reference Manual* for details.). [end]

Rationale: To check the integrity of the FIRC

Note

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

5.2.8.2 Runtime checks

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

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

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

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

5.2.9 Fast External Oscillator (FXOSC)

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

5.2.9.1 Initial checks and configurations

1. Nominal frequency of the FIRC is 16 MHz, but the post trim accuracy over voltage and temperature must be taken into account (see the *MPC5748G Data Sheet*).

Assumption:[SM_175] FlexRay and FlexCAN, both of which feature modes to be clocked directly by the FXOSC, should not make use of these modes in normal operation unless effects of clock glitches are sufficiently detected by the applied FT-COM layer. [end]

5.2.9.2 Runtime checks

Assumption: [SM_176] Software shall check that the system clock is available, and sourced by the FXOSC, before running any safety relevant function or enabling the FCCU into the operational state.[end]

5.2.10 PLL Digital Interface (PLLDIG)

The MPC5748G uses a FMPLL (frequency modulated clock) used to generate high speed clocks. The FMPLL provides a loss of lock error indication that is routed to the FCCU (see the 'FCCU error inputs' table for particular fault channel). If there is no PLL lock, the system clock can be driven by the FIRC. Glitches which may appear on the crystal clock are filtered by the FMPLL (low-pass filter).

Implementation hint: PLLDIG_PLLSR[LOLF] indicates that a loss of lock event occurred. The PLLDIG_PLLCR[LOLIE] can be set to enable an interrupt request upon loss of lock.

5.2.10.1 Initial checks and configurations

After system reset, the external crystal oscillator is powered down and the PLL is deactivated. Software shall enable the oscillator. The MPC5748G uses after system reset the Fast Internal RC Oscillator (FIRC) as clock source (see the "Oscillators" chapter in the *MPC5748G Reference Manual* and Fast Internal RC Oscillator (FIRC) for details on FIRC configuration).

Assumption: [SM_178] Before executing any safety function, a high quality clock (low noise, low likelihood for glitches) based on an external clock source shall be configured as the system clock of the MPC5748G. [end]

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

Implementation hint: The PLL can be configured to use the Fast External Oscillator (FXOSC) or the Fast Internal RC Oscillator (FIRC) as clock reference. In general MC_CGM_AC5_SC[SELCTL] shall be set to 1 to select the FXOSC (reset value is 0).

Assumption under certain conditions:[SM_179] When clock glitches endanger the system level functional safety integrity measure, respective functional safety-relevant modules shall use the PLL as clock source. 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 MPC5748G.

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

During/after initialization but before executing any safety function, application software has to check that the MPC5748G uses the PLL clock as "system clock".

Implementation hint: Application software can check the current system clock by checking the MC_ME_GS[S_SYSCLK] flag. MC_ME_GS[S_SYSCLK] = 2 indicates that the FMPLL clock is being used as the system clock.

5.2.11 Clock Monitor Unit (CMU)

At startup, the CMU is not initialized and the FIRC is the default system clock.

Stuck at faults on the external oscillator are not detected by CMU at power-on since the monitoring unit are not initialized and the MCU is still running on the FIRC. It is the responsibility of the application software to check that the system clock is available and based on the FXOSC before running any safety element function or setting the FCCU into "operational" state.

Clocks are supervised by Clock Monitoring Unit (CMU). The CMU is driven by the FIRC to ensure independence from the monitored clocks. CMU flag errors associated with conditions due to clock out of a programmable bounds and loss of reference clock. If a supervised clock leaves the specified range for the device, an error signal is sent to the FCCU. MPC5748G includes the CMU shown in Table 5-1.

Table 5-1. Clock Monitoring Unit

CMU	Monitored Clock
CMU	Loss of PLL or FXOSC

The CMU uses the FIRC as the reference clock for independent operation from the monitored clocks. The purpose is to check for error conditions due to:

- Loss of clock from external crystal (FXOSC)

- PLL clock out of a programmable frequency range (frequency too high or too low)
- Loss of PLL clock

The CMU supervises the frequency range of the clock source. In case of abnormal behavior, the information is forwarded to the FCCU as faults (see FCCU mapping of faults).

Assumption: [SM_180] For safety-relevant applications, the use of the CMU is mandatory. If the modules that the CMU monitors are used by the application safety function, the user shall verify that the CMU is not disabled and its faults are managed by the FCCU. The FCCU's default condition does not manage the CMU faults, so it must be configured accordingly. [end]

5.2.11.1 Initial checks and configurations

Assumption: [SM_181] 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 CMU should be used for clock monitoring. Application software shall check that the CMU is enabled, and its fault managed by the FCCU.

Implementation hint: In general, the following application dependent configuration shall be executed before CMU monitoring can be enabled:

- Software configures CMU_CSR[RCDIV] to select a FIRC divider. The divided FIRC frequency is compared with the FXOSC.

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

5.2.12 Mode Entry (MC_ME)

Assumption under certain conditions:[SM_182] 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 device. This unit includes the internal regulator for the logic power supply (1.25 V) and a set of voltage monitors. Particularly, it embeds low voltage detectors (LVD) and high voltage detectors (HVD). If one of the monitored voltages drops below the LVD threshold, a power on reset is generated. If the monitored voltages exceeds the HVD threshold, a functional reset will be generated. These reactions are initiated to control erroneous voltages before these cause a CMF (for correct operating voltage ranges please see the *MPC5748G Data Sheet*).

To ensure functional safety, the Power Monitoring Controller (PMC) monitors various supply voltages of the MPC5748G device:

- The low and high voltage detectors (LVD/HVD) supervise the 1.25 V core supply (VDD_LV) voltage to verify that it maintains a level between the lower and upper limits.
  - LVD_VDD
  - HVD_VDD[2]
- LVD detects voltages that fall below specified values as shown in the *MPC5748G Data Sheet*.

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

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

Undervoltage and overvoltage conditions are primarily reported to the MC_RGM, where they directly cause a transition into a safe state with a power on reset or functional reset, resepectively. This solution was chosen because safety-relevant voltages have the potential to disable the failure indication mechanisms of the MPC5748G (the FCCU). The FCCU error reporting is not utilized since HVD errors are handled by the MC_RGM.

Note

Only for development purposes, different fault reactions can be programmed in the PMC for HVD error reporting to disable the MC_RGM reset capabilities.

2. Abbreviations used in the "Power Management Controller (PMC)" chapter of the *MPC5748G Reference Manual*.

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

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

5.2.13.1 1.25 V supply supervision

Voltage detectors LVD_VDD and HVD_VDD 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 MPC5748G, a POR 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_189] When the system requires robustness regarding 1.25 V over voltage failures, the external VREG mode is preferably selected. The internal VREG mode uses a single pass transistor and, therefore, 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. Alternatively, the digital (1.25 V) core supply voltage may be monitored externally and the power supply shut-down in case of an over voltage.

5.2.13.2 3.3 V supply supervision

Voltage detectors LVD_VDD_IO_A_LO and LVD_VDD_FLA monitor the 3.3 V supply 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 MPC5748G, a POR is triggered.

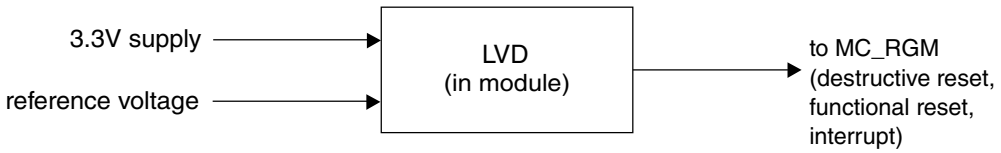

Figure 5-2. Logic scheme of the 3.3 V voltage detectors

5.2.13.3 5 V supply supervision

Voltage detector LVD_IO_A_HI monitors the 5 V supply for under voltage in relation to a reference voltage. The figure below depicts the logic scheme of the voltage detector. In case a single LVD detects under voltage during normal operation of the MPC5748G, a POR is triggered.

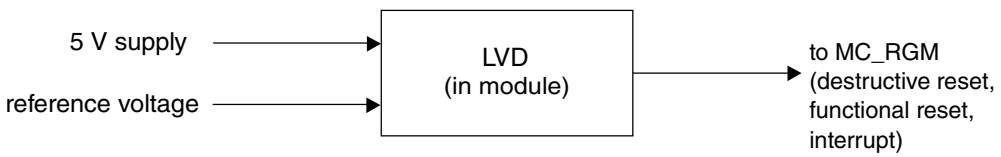

Figure 5-3. Logic scheme of the 5 V voltage detectors

5.2.14 Memory Protection Units

As a multi-master, concurrent bus system, the MPC5748G provides safety mechanisms to prevent non-safety masters from interfering with the operation of the core. MPC5748G also contains mechanisms to handle the concurrent operation of software tasks with different or lower ASIL classifications.

Recommendation: For ASIL B applications, the SMPU 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 System Memory Protection Unit (SMPU)

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

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

Assumption: [SM_194] The SMPU shall only be programmed by the 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 main core. [end]

5.2.14.2 Initial checks and configurations

Assumption under certain conditions: [SM_195] When bus masters are used (for example, FlexRay), system level functional safety integrity measures must cover bus operations to reduce the likelihood of said resources being erroneously modified. [end]

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

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

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

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

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

5.2.15 Peripheral Bridge (PBRIDGE) protection

The Peripheral Bridge (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 safety-relevant 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 before exiting reset (destructive reset or external reset).

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

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

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_197] After startup and before the safety application starts, application software shall confirm all LBISTs and MBISTs finished successfully and no further errors are flagged. [end]

Assumption:[SM_209] 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]

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_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.

3. This does not include flash memory.

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 a safety integrity measure to detect latent faults.

The MPC5748G is equipped with a Built-in hardware self-test:

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

Boot-time tests (MBIST, LBIST) are performed after the occurrence of a destructive or external reset, unless they are disabled. All boot-time tests are executed before application software starts executing. If failed, the MCU 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 NXP) 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 Memory Built-In Self-Test (MBIST)

The SRAM BIST (MBIST) runs during initialization (during boot).

NOTE

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

5.2.16.2 Logic Built-In Self-Test (LBIST)

The Logic Built-In Self-Test (LBIST) runs during initialization (during boot).

NOTE

A destructive reset should be triggered at least once per L-FTTI (for example, once per drive cycle) to ensure LBIST is performed.

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 ECC logic check

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

5.2.16.5 Flash memory ECC fault report check

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

5.2.17 End-to-end ECC (e2eECC)

The MPC5748G includes end-to-end ECC (e2eECC) support for improved functional and transient fault detection capabilities. Memory-protected by the traditional ECC/EDC generates and checks additional error parity information local to the memory unit to detect and/or correct errors which have occurred on stored data in the memory.

In contrast, in the MPC5748G e2eECC protected memory, the bus master initiates the data write and generates ECC checkbits based on 29-bit address and 64-bit data fields. The data including the checkbits are transferred from the bus master to the appropriate bus slave. Both data and checkbits are stored into the memory. When the bus master initiates a read of previously written memory location, the read data and checkbits are passed onto the system bus interconnection. The bus master captures the read data and associated checkbits, performs the ECC checkbit decode and syndrome generation and performs any needed single-bit correction.

The e2eECC provides:

- ECC for master-slave accesses via the crossbar
- ECC is stored in the memories on write operations and validated by the crossbar master on every read operation
- 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, FlexRay RAM array.
- - ECC on address and data
  - SECDED covers 64-bit data and 29-bit address bits

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

The ECC used for flash memory marks All-0 as being in error, but allows All-1 situations to take into consideration reading erased, uninitialized flash memory.

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

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

5.2.18 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_198] 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 SWT can be configured to start when the interrupt request is generated and the application software can read the timer value to determine when the ISR is entered. This method can be used to determine whether the measured interrupt latency exceeds the requirements.

Assumption:[SM_199] Periodic low latency IRQs will use a running timer/counter to ensure their call period is expected.[end]

5.2.18.2 Non-Periodic low latency IRQs

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

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_300] 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.

Failures outside of the eDMA can lead to faulty eDMA operation, such failures have to be detected by software.

5.2.19.1 Runtime checks

Assumption:[SM_301] The eDMA will be supervised by software which detects spurious, too often, or constant activation.[end]

Rationale: Prevent the eDMA from stealing transfer bandwidth on the AXBS, 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 BCTU and ADC, the number of the converted ADC channels is saved into the BCTU FIFO together with the acquired value. The eDMA transfers this value from the BCTU 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_202] Applications that are not resilient to spurious, or missing functional safety-relevant, eDMA requests can not use the PIT module to trigger functional safety-relevant eDMA transfer requests. [end]

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

5.2.19.1.1 eDMA transfers

In cases where the eDMA is used to transfer data (for example, to 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_304] If safety-relevant software is using the eDMA to transfer data to a peripheral or the RAM, the following holds:[end]

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

5.2.20 System timer module

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.

Assumption: [SM_205] 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_206] The STM IRQ handler shall be under SWT protection.[end]

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

5.2.21 Periodic interrupt timer

5.2.21.1 Runtime checks

Assumption: [SM_207] When using PIT module, the PIT module should be used in such a way that a possible functional safety-relevant failure is detected by the Software Watchdog Timer (SWT). [end]

Rationale: To catch possible PIT failures

Recommendation under certain conditions: [SM_208] If the PIT is used by the application software in a safety function, a checksum of its configuration registers using the CRC must be calculated and compared with the expected one to verify that the PIT configuration is correct. [end]

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

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

5.2.22 System Status and Configuration Module

5.2.22.1 Initial checks and configurations

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

Implementation hint: After reset, the BAF is automatically executed; however, once the BAF has finished it branches to RAM and disables the BAF region so that it can no longer be executed by the core. The PFLASH_PFCR3[BAF_DIS] field controls executable access to the BAF (Boot Assist Flash) region of the flash. Once this field is set, attempted instruction accesses targeting the BAF region are aborted and terminated with a system bus error. Data-type accesses to the BAF region are not affected by this field. Once this field is set, it becomes a read-only field and can only be cleared by hardware reset, and any subsequent write attempts to modify this field are ignored with an error-free data transfer termination.

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 the MEMU collects are stored in reporting tables that are accessible through the MEMU register interface. The application software can write known error addresses into the MEMU reporting table to prevent redundant reporting of those errors to the FCCU in the event the same addresses are accessed again.

5.2.24 Flash memory

The MPC5748G provides 6 MB of programmable non-volatile (NVM) flash memory with ECC which can be used for instruction and/or data storage.

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

5.2.24.1 EEPROM

The MPC5748G provides blocks of flash memory for EEPROM emulation. ECC events detected on accesses to the EEPROM flash memory blocks are reported to the Memory Management Unit (MEMU). Single-bit errors are corrected and signaled to the MEMU. Multi-bit errors are replaced by a fixed word (representing an illegal instruction), and are 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.

Assumption: [SM_212] Before executing any safety function, a flash memory array integrity self check should be executed. The calculated MISR value is dependent on the array content and therefore has to be validated by system level application software.[end]

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

Implementation hint: This test may be started by application software: its result may be validated by reading the corresponding registers in the flash memory controller after it has been finished (see "Array integrity self check" section in the "Flash memory" chapter of the *MPC5748G 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_216] According to the specific Flash usage by the application software, a software test should be implemented to check for potential multi-bit errors introduced by permanent failures in Flash control logic (e.g. read pump, read timing, Vref, etc.).[end]

Assumption: [SM_217] A SW safety mechanism shall be implemented to ensure the correctness of any write operation to both Flash and 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_219] Flash memory ECC failure reporting path should be checked to validate if detected ECC faults are correctly reported. [end]

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

Implementation hint: 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 MPC5748G Reference Manual) a set of test patterns to test the integrity of the ECC logic fault reporting path to the MEMU and FCCU (executed at start-up, latent failure measure).

5.2.25 Body Cross Triggering Unit (BCTU)

The ADC BCTU allows automatic generation of ADC conversion requests with minimal CPU intervention. The BCTU generates some triggers based on input events from eMIOS and PIT.

The BCTU can be used if the application needs to synchronize the reading of some ADC inputs with MCU events (for example, PWMs and/or timers).

5.2.25.1 Runtime checks

Assumption:[SM_220] 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, a set of ADC commands and pulses to be generated can be defined.

If the application safety function includes reading of synchronized inputs with events (for example, PWMs, timers, or any combination), the system integrator can use the BCTU module for this purpose. The required software needed is listed in Synchronize sequentially read inputs.

For a detailed description of BCTU operation (triggered and sequential mode), its configuration, and use, see the MPC5748G Reference Manual.

5.2.25.2 Synchronize sequentially read inputs

Assumption:[SM_221] If the BCTU is part of the application safety function, the safety integrity is achieved by a mix of hardware mechanisms and software safety integrity functions implemented at the application level, examples of which are: [end]

- Example: BCTU ADC conversion data overrun test
- Example: BCTU ADC channel cross-check
- Example: BCTU timer command cross-check

NOTE

These functions are mandatory only if the BCTU is used in the application.

5.2.25.2.1 Example: BCTU ADC conversion data overrun test

In case new ADC conversion data is available before the old data has been read, the old data will be over-written, an overrun indication flagged, and the application software may have to handle the error indication.

Rationale: Tests if all the triggers configured within a control period have had their ADC conversion data read.

Implementation hint: The Body Cross Triggering Unit Module Status register (BCTU_MSR) shows information about the overrun status. When the BCTU detects an overrun error, an interrupt is generated.

5.2.25.2.2 Example: BCTU ADC channel cross-check

The BCTU stores in registers BCTU_ADC0DR and BCTU_ADC1DR both the value provided by each ADC conversion and the channel number. Application software checks the ADC channel number sequence against what is expected.

Rationale: To detect if an incorrect channel has been acquired.

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

5.2.25.2.3 Example: BCTU timer command cross-check

Application software configures a timer channel (eMIOS_*n*) to count the number of timer commands generated within a BCTU control period and checks the number against the expected count.

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

5.2.26 Error reporting path tests

It is possible to use fault injection to check the correct operation of several reporting paths from supervisors to the FCCU. The FCCU input table specifically lists those inputs in the table “FCCU non-critical fault mapping” in the *MPC5748G Reference Manual*.

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

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

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

A multiple cell failure caused for example, by a neutron or alpha particle or a short circuit between cells may cause three or more bits to be corrupted in an ECC-protected word. As result, either the availability may be reduced or the ECC logic may perform an additional data corruption labeled as single-bit correction. This is prevented within the design of MPC5748G 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. MPC5748G 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 MPC5748G. A selectable (WKPU_NCR[NFE0]) analog glitch filter is implemented on the NMI-input. 30 external wake-up sources can be configured to have an analog filter. 32 external interrupt sources 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, wake-up, or interrupt inputs.

5.2.28 Register Protection module (REG_PROT)

The Power 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 MPC5748G is protected from errant register changes made by other user-mode routines. User versus supervisor mode can also be used as a decision criteria in the MPUs and the peripheral access control (PAC) of the PBRIDGES.

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

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

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

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

If registers are protected against random bit flips they shall also be protected against accidental SW writes (SW register protection).

5.2.28.1 Runtime checks

Assumption: [SM_225] For safety relevant applications, all configuration registers, and registers that aren't modified during application execution, must be protected with a Hard Lock. [end]

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

Implementation hint: Register Protection address space is inside the memory space reserved for the peripherals (see the "Register protection configuration" section of the *MPC5748G 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_226] If external NMI and Wake-up are used as a safety mechanism, especially if waking up within a certain timespan or at all is considered safety-relevant, it is required to implement corresponding system level measures to detect latent faults in the WKPU. [end]

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

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

5.2.30 Crossbar Switch (AXBS)

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

Assumption: [SM_227] Masters of the AXBS which are not safety-related shall have a lower arbitration priority on the AXBS compared to safety-related masters. [end]

5.2.30.1 Runtime checks

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

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

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

Assumption: [SM_228] Within the FTTI, application software shall detect failures of the AXBS configuration affecting system performance by using the configuration readback and SWT monitoring described above.[end]

5.2.31 System Integration Unit Lite2 (SIUL2)

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

Also, intelligent self-checking sensors are possible if the data transmitted from the sensors contains redundant information in the form of a checksum, for example. Preferably, the replicated modules generate or receive the replicated data using different coding styles (for example, inverted in the voltage domain or using voltage and time domain coding for redundant channels). Safety system developers may choose the approach that best fits their needs.

Assumption: [SM_233] 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 involve using 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 modules (for example, modules configured for PWM or timer operation) to implement multiple independent and different channels.

5.2.31.1 Digital inputs

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

5.2.31.2 Hardware

Implementation hint: Digital inputs used for functional safety may need to be acquired redundantly. To reduce the risk of CMFs, the redundant channels may not use adjacent GPIOs (see Causes of dependent failures).

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

5.2.32 Analog-to-Digital Converter (ADC)

Parts of the Successive Approximation Register (SAR) Analog-to-Digital Converters (ADCs) of the MPC5748G do not provide the functional safety integrity to achieve high functional safety integrity targets. Therefore, system level safety measures are required.

5.2.32.1 Initial checks and configurations

Assumption under certain conditions:[SM_130] When the Analog-to-Digital Converter (ADC) of the MPC5748G are used in a safety function, suitable system level functional safety integrity measures must be implemented after reset (external reset or destructive reset) before starting the respective safety function to ensure ADC integrity. [end]

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

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

5.3 Communications

5.3.1 Redundant communication

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

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

- Deserial Serial Peripheral Interface (DSPI)
- LINFlexD Communication Controller

These communication protocols do not contain special functional safety mechanisms other than what is included in their protocol specifications. The system level communication architecture needs to provide the functional safety mechanisms on the interface of the modules to meet functional safety requirements.

5.3.2 Fault-tolerant communication protocol

Parts of the integrated FlexRay, LINFlexD, FlexCAN, DSPI/SPI, I2C, SAI/I2S, MLB150, USB, and Ethernet communication channels do not on their own provide the functional safety integrity that the IEC 61508 series and ISO 26262 require for high functional safety integrity targets.

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

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

FlexRay, FlexCAN, and LINFlexD do not have specific functional safety mechanisms other than ECC protection of SRAM arrays and what is included in their protocol specifications. 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

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

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.

5.4 Additional configuration information

5.4.1 Stack

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

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

5.4.1.1 Initial checks and configurations

Assumption under certain conditions:[SM_103] When stack underflow and stack overflow due to systematic faults within the application software 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: Core debug support features software facilities that can be used for stack limit checking when not used for debugging. The DAC1 and DAC2 resources maybe 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 EDM or IDM mode, and when resources used for stack limit checking are owned by software, will utilize a DSI or machine check exception.

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

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

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

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

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

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

Note

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

5.4.2 MPC5748G configuration

Assumption: [SM_240] It is required that application software verifies that the initialization of the MPC5748G is correct before activating the safety-relevant functionality.[end]

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

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 MPC5748G'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 MPC5748G's safety function or disable the safety mechanism due to their change.

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

Chapter 6
Failure Rates and FMEDA

6.1 Failure rates

In order to analyze and quantify the effectiveness of the MPC5748G integrated safety architecture to handle random hardware failures, the inductive analysis method of FMEDA (Failure Modes Effects and Diagnostic Analysis) was performed during the development of the MPC5748G. The following methods for deriving the base failure rates of the MPC5748G were used as input to the FMEDA:

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

6.2 FMEDA

In order to support the integration of the MPC5748G into safety-related systems and to enable the safety system developer to perform the system level safety analysis, the following documentation is available:

- FMEDA - Inductive analysis of the MPC5748G enabling customization of system level safety mechanisms, including the resulting safety metrics for ISO 26262 (SPFM, LFM and PMHF) and IEC 61508 (SFF and β-factor β_{IC})
- FMEDA Report - Describes the FMEDA methodology and safety mechanisms supported in the FMEDA, including source of failure rates, failure modes and assumptions made during the analysis.

The FMEDA and FMEDA report are available upon request.

6.2.1 Module classification

For calculating the safety metrics for ISO 26262 (Single-Point Failure Metric (SPFM), Latent Failure Metric (LFM) and Probabilistic Metric for random Hardware Failures (PMHF)) and for IEC 61508 (Safe Failure Fraction (SFF) and $ß_{IC}$ factor) the modules of the MPC5748G 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 MPC5748G can be found in the attached "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 MPC5748G on chip level:

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

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

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

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

7.1.2 Measures against dependent failures

7.1.2.1 Environmental conditions

7.1.2.1.1 Temperature

The MPC5748G was designed to work within a maximum operational temperature profile (see the *MPC5748G Data Sheet*).

7.1.2.1.2 EMI and I/O

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

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

7.1.2.2 Failures of common signals

7.1.2.2.1 Clock

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

7.1.2.2.2 Power supply

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

7.1.2.2.3 Nonapplication control signals

Modules and signals (for example, for scan, test and debug), which are not safety-related should never be able to lead to a safety-related failure. This can be ensured by either not interfering with the safety-related parts of the MPC5748G or by detecting such interference. For example, there must be assurance that the system is not debugged (or unintentionally placed in debug mode), or placed in any other special mode different from normal application execution mode (for example, test mode). In addition, an FCCU failure indication is generated if:

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

7.1.3 Dependent failure avoidance on system level

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

- Usage of non-contiguous balls of the package
- Usage of non-contiguous pads of the silicon
- Usage of peripheral modules not sharing the same PBRIDGE
- Non-contiguous routing of these signals on the PCB

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

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

The pin/ball assignment for individual peripherals can be extracted from the *MPC5748G Microcontroller Data Sheet*. The following section explains how this can be achieved.

7.1.3.1 I/O pin/ball configuration

Whether two functions on two signals are adjacent to each other can be determined by looking at the mechanical drawings of the packages (see the *MPC5748G Data Sheet*) together with the ball number information of the packages as seen in the *MPC5748G Reference Manuals* "System Integration Unit Lite2 (SIUL2)" section and the "Pin muxing" table.

The layout of the device balls and the order of die pad signals need to both be taken into consideration. Adjacency of the package balls is straight forward since it can be seen in the package layout. It is more difficult to determine adjacency on the die. The Signal Description chapter in the MPC5748G Reference Manual can be used in assisting to determine adjacency of signals on the die. To help avoid potential issues, redundant signals cannot be on adjacent balls or on adjacent die pads. Avoiding adjacency limits crosstalk, signal drive strength, and other associated issues.

7.1.3.2 Modules sharing PBRIDGE

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

7.1.3.3 External timeout function

A dependent failure may lead to a state where the MPC5748G is not able to signal an internal failure via its FCCU_EOUT*n* signals (error out). With the use of a system level timeout function (for example, watchdog timer), the likelihood that dependent failures affect the functional safety of the system can be reduced significantly.

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

- General destruction of internal components (for example, due to non-mitigated overvoltage or a latch-up at redundant input pads). Since these errors do not result in subtle output variations of the MPC5748G but typically in a complete failure, a simple watchdog is sufficient.

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

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

NOTE

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

The external watchdog function is in permanent communication with the CPU of MPC5748G. As soon as there are no correct communications, the external watchdog function switches the system to Safe $state_{system}$. Thus, either the MPC5748G or external watchdog function can transition the system to Safe $state_{system}$. The external watchdog function is required to be sufficiently independent of the MPC5748G (for example, regarding clock generation, power supply, and so on).

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

7.1.4 β_{IC} considerations

During the development of the MPC5748G, the susceptability of the MCU to dependent failures is evaluated by ensuring sufficient independence between on-chip functions and their respective safety mechanisms.

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

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

Chapter 8
Additional Information

8.1 Testing All-X in RAM

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

8.1.1 Candidate address for testing All-X issue

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

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

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

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

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

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

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

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



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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

Example:

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

RAM base address = 40000000h

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

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

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

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

RAM base address = 40000000h

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

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

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

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

8.1.2 ECC checkbit/syndrome coding scheme

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

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

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

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

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

Table 8-1. e2eECC basic H-matrix definition

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

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

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

Figure 8-1. C Language encode ECC function description

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    return(ecc);
}
```

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

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

Chapter 9
Acronyms and Abbreviations

9.1 Acronyms and abbreviations

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

Table 9-1. Acronyms and abbreviations

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

Table continues on the next page...

Table 9-1. Acronyms and abbreviations (continued)

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

Appendix A
Release Notes for Revision 3

A.1 General changes

- Editorial changes and improvements throughout this document.
- Updated the attached "Module classification" spreadsheet (release notes are contained within the attachment).
- Changed the name of the module classification spreadsheet attachment to "Module Classification".

A.2 Preface changes

- In Functional safety standards, changed "ISO 26262-10 Annex A ISO 26262 and microcontrollers" to "ISO 26262-10:2011-2012 Annex A ISO 26262 and microcontrollers".
- In Other considerations, changed "The safety system developer" to "The functional safety manager for the developed and deployed system".

A.3 MCU Safety Context changes

- Cleaned up Figure 2-1.
- In the Faults and failures section:
  - Modified the hierarchy of this section - no content changes.

- In the MCU fault indication time section:
  - Removed the list of mechanisms from the "Recognition time" bullet.

A.4 MCU Safety Concept changes

- In Figure 3-1 :
  - Added a note: "All FlexCANs optionally support CAN FD'.

- In External error indication :
  - Added the following note to the end of the section: "EOUT does not indicate the fault condition if the MCU is in RESET. When the MCU is in RESET and the IO is high-z, the system safe state should be assured using pull-up/down resistors to pull EOUT to its fault indication level."

- In ECC :
  - In the introductory paragraph "Error correcting codes are used for end-to-end protection...", changed "cores" to "bus masters" and "RAMs" to "RAMs and PBRIDGES a and b".
- In End-to-End protection on data path :
  - Updated Figure 3-2.
- Replaced the ECC for storage section.
- In Communication controllers :
  - Added Ethernet, MediaLB, uSDHC, and USBOTG to the list of communication controllers that do not contain special safety mechanisms.
- In Disabling of communication controllers :
  - Added a note at the end of this section: "The FCCU uses internal signals to disable..."
- Replaced the BIST during boot section.
- Added the LBISTed modules section.
- In External error indication :
  - Added the note "FCCU EOUT0 and EOUT1 are muxed on the pad, and their controls will be via SIUL. There will be no interaction with the HSM for this control."
  - Changed "FCCU_STAT[PhysicErrorPin]" to "FCCU_STAT".
- In Fault inputs :
  - Changed the referenced chapter from "Chip Configuration" to "Fault Collection and Control Unit (FCCU)".
- In Operational interference protection :
  - Added the following sentence at the end of this section: "These safety mechanisms are further described in the SMPU chapter of the MPC5748G Reference Manual."
- Added the Common cause failure measures section.
- In FCCU and failure monitoring :
  - Removed the "FCCU supervision (FOSU)" subsection.

- In the Built-In Self Tests (BIST) section:
  - Removed the subsection "Online Logical BIST (LBIST)".
- In the BIST during boot section:
  - Added the following paragraph at the end of the section: "A destructive reset should be triggered at least once per L-FTTI (for example, once per drive cycle) to ensure an offline LBIST is performed. In some applications, the MPC5748G may not be reset within the L-FTTI but may instead enter Low Power mode within the L-FTTI. In this case, a destructive reset can be triggered upon exit from Low Power mode, triggering an offline LBIST."

A.5 Hardware Requirements changes

- In the Error Out Monitor (ERRM) section:
  - Changed "FCCU_EOUT0, and optionally FCCU_EOUT1" to "FCCU_EOUT0 and/or FCCU_EOUT1".

- In the Power Supply Monitor (PSM) section:
  - In Assumption SM_087, removed the phrase "where no supervision is provided on the MCU".

- In the Single FCCU signal connected to separate device using voltage domain coding section:
  - Changed "SM_170" to "SM_170a".

A.6 Software Requirements changes

- In the EEPROM section:
  - Corrected the first paragraph to show that ECC events detected on accesses to EEPROM, single-bit errors, and multi-bit errors are all reported to the MEMU.

- In the Test mode section:
  - Added the following sentence at the end of the section: "FIRC clock-related test mode activation is intended to be covered by the frequency meter function of CMU_0, as described in the FIRC Runtime checks section."

- In the CRC Runtime checks section:
  - Changed "SM_170" to "SM_170b".
- In the Built-in Hardware Self-Tests (BIST) section:
  - Changed the following sentences to be Assumption SM_209: "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."
  - Removed the following paragraph at the end of the section: "If additional coverage of the ECC error reaction path is requested, SW shall program two patterns into RAM..."

- In the STM Runtime checks section:
  - Removed Assumption SM_310.
  - Added the Implementation hint.
- In the 5 V supply supervision section:
  - Added this new section.
- In the Fast Internal RC Oscillator (FIRC) section:
  - Added a sentence stating that the FIRC should not be used as the input of the PLL for the system clock.
- In the Built-in Hardware Self-Tests (BIST) section:
  - Removed the statement that the STCU2 will execute automatically "when initiated by software (online)".
  - Removed STCU_LBSSW and STCU_LBESW checks of LBIST results.
  - Removed STCU_MBSLSW, STCU_MBSMSW, STCU_MBSHSW, STCU_MBELSW, STCU_MBEMSW, and STCU_MBEHSW checks of MBIST results.
- In the Memory Built-In Self-Test (MBIST) section:
  - Removed the statement that the SRAM BIST (MBIST) "can be run during shutdown, if configured appropriately and triggered by software".
  - Removed the reference to the STCU2 section.
- In the Logic Built-In Self-Test (LBIST) section:
  - Removed the statement that the Logic Built-In Self-Test (LBIST) "can be triggered by software run during shutdown if configured appropriately".
  - Removed the reference to the STCU2 section.
  - Removed the note "In principle, the LBIST can be run at any time, but a reset will be generated by the MCU after the LBIST completes."
  - Added the note "A destructive reset should be triggered at least once per L-FTTI (for example, once per drive cycle) to ensure LBIST is performed."

A.7 Failure Rates and FMEDA changes

- In the Module classification section:
  - Changed the name of the module classification spreadsheet to "Module Classification".

A.8 Dependent Failures changes

- No substantial content changes

A.9 Additional Information changes

- In the ECC checkbit/syndrome coding scheme section:

- Added the following footnote to the "Data Bit" heading (applies to both the "Data Bit" and "Address Bit" headings) in Table 8-1 : "Bit numbering is AHB convention, bit 0 is LSB. D[7:0] corresponds to byte at address 0. D[63:56] corresponds to byte at address 7."

A.10 Acronyms and Abbreviations changes

- No substantial content changes

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number MPC5748GSM
Revision 3, 08/2017