White Paper

Using Decoupled Parallel Mode for Safety Applications

Author: Markus Baumeister

Document Number: MPC564XLWP
REV 0

freescale™ semiconductor

Content

1 Introduction

Over the last few years, automotive electronic systems have become a dominant factor in defining the driving experience of modern vehicles. Increasingly, automotive electronic systems need to fulfill functional safety requirements not only in active and passive safety systems, but also in chassis, powertrain and body applications. In this context, functional safety is often considered as the part of the overall safety relating to the equipment under control (EUC) and the EUC control system that depends on the correct functioning of the electronic system.

To support the requirements of automotive functional safety applications, Freescale has introduced the dual-core MPC564xL MCU family. This family contains two “channels” that each consist of a core, bus, interrupt controller, memory controller and other core-related modules. The channels[1] can operate in either one of two distinct operating modes: a Lockstep Mode (LS Mode or LSM) and a Decoupled Parallel Mode (DP Mode or DPM).

In LSM, the two channels of the MCUs operate in lockstep. Any deviation in the output of the two channels is detected by hardware and signaled as a possible failure (see [2] Figure 1 and Section 5.1). The LSM provides a high diagnostic coverage and short detection intervals on a hardware level without software interaction. This capability comes at the cost of a higher computational overhead since the two cores provide the performance of only one.

In DPM, the two channels of the MCU work independently. Automatic hardware checks for equal operation between the two channels are not executed although some master/checker combinations orthogonal to the channels will continue to check for correct hardware operation. Thus the replicated components can be used to improve performance but will not provide any diagnostic coverage at the hardware (HW) level without significant software (SW) interaction.

The choice between operating the MCU in LSM or DPM is often driven by the tradeoff between performance and SW transparency. In some automotive systems, it is desirable to have the higher performance of the DPM for non-safety-related functions and to execute some safety-relevant functions that do not require high performance. In this case, the question is how to achieve safety in DPM.

In general, the answer to this question depends on the application concept and software implementation. This paper[2] provides suggestions how DPM can be utilized and describes additional measures that are necessary to compensate for the lack of hardware level checking that is provided in LSM.

In Chapter 2, techniques based on architecture examples show how software can cover the self crosscheck for correct operation and substitute the functionality of the hardware checkers in LSM.

Chapter 3 lists specific aspects for executing safety-relevant software in DPM while Chapter 4 focuses on safety measures to be applied by the user in (MPC564xL’s) LS Mode that are still necessary in DP Mode. Chapter 5 explains which hardware safety measures implemented for LSM still work in DPM. Finally, Chapter 6 compares the two modes.

[1] In this paper, we will often refer to the channels only as cores since those are the major components of the channels and provide a more intuitive terminology.

[2] Note that this paper is not backed by a rigorous Failure Modes, Effects and Diagnostic Analysis (FMEDA) or similar analysis: therefore no guarantee can be given on its completeness. As the number of variants mentioned in this paper shows, such an analysis would be difficult without a specific application background.

2 Application Architectures

With two cores executing independently on one MCU, there are several software architectural options feasible to achieve functional safety. In this chapter, a selection of possible assignments of safety-related (SR) and non-safety-related (NSR) software functions to the cores are discussed and some examples are given on how to access sensors and actuators.

2.1 Core Usage

In contrast to the LSM, the DPM provides no inherent protection at the hardware level against random hardware faults[1]. Therefore, in DPM, some type of redundancy has to be included at the software level. The hardware redundancy in LSM always covers the complete, symmetrically redundant execution with an automatic check for equality. In DPM, redundancy at the software level is not fixed but can be adapted to the specific requirements of the individual applications at the cost of a recurring implementation effort for every new application.

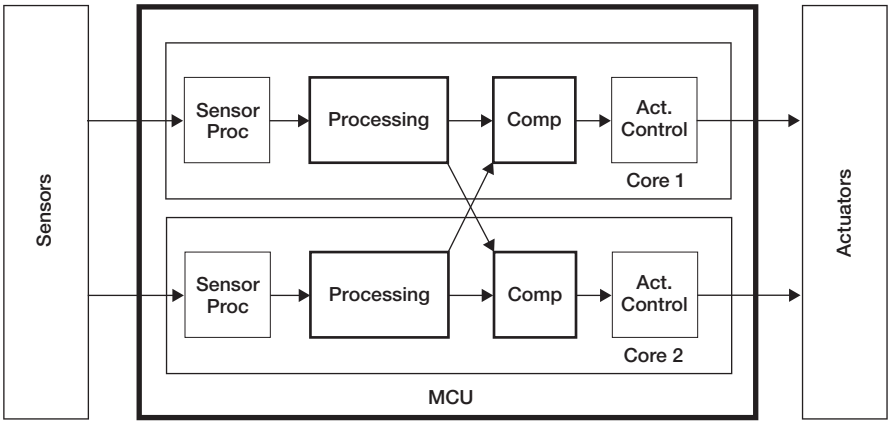
As shown in the following examples, functions can be distributed onto the two independent cores in DPM to achieve the software level redundancy mentioned above. In these examples, the connections to sensors and actuators are shown for illustrative purposes. (Other connection types shown in sections 2.2 and 2.3 are also feasible).

2.1.1 Symmetric Redundancy

In a symmetrical redundancy approach, two instances of the safety-relevant software are executed on the cores using the same or very similar inputs. The results are then compared in software, either for equality or for a sufficiently small deviation, and forwarded to the actuators only if they are within the acceptable deviation limits. To avoid failures of one core causing simultaneous wrong results and disabling the tests, the comparison is normally executed on both cores. This is shown in Figure 1, which also shows sensors, actuators and the software components for reading and/or writing from both cores. Those components are the topic of later sections and might consist of more than simple driver calls due to the need to synchronize access times and values between the cores (see [4]).

If the results in this approach are compared for equality, it is the software equivalent of the LSM albeit with a much higher comparison granularity (dedicated end results instead of each intermediate result) and a longer comparison latency.

Figure 1 Symmetric Redundant Processes



This software architecture makes most sense if a significant amount of non-safety-related software needs to be executed and/or if the safety-related software was implemented with a methodology such as n-version programming (see [1] as well as [5] p.207ff) to mitigate the risk of software faults. The main advantage of this approach is then the comparatively low software effort and its simple structure.

But if the same software is replicated on both cores and no additional non-safety related software is executed, the main disadvantage of this approach is that the performance is worse than in LSM. Not only is the software executed twice (two cores just provide the performance

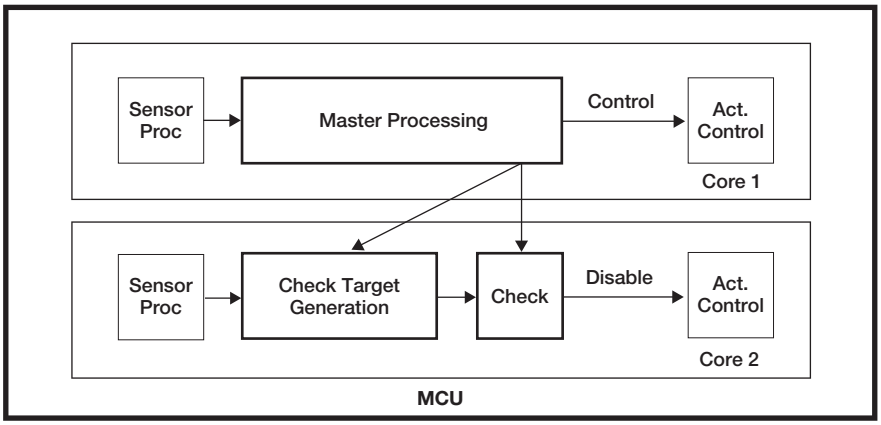
[1] See [5] Chapter 2 for an overview of terminology concerning faults, errors, failures, etc.

of one) but in contrast to LSM, this happens in a non-transparent way, so variables are stored twice in RAM[1] requiring space and bus bandwidth. Additionally, the comparisons that are performed in hardware in LSM need to be implemented and executed in software, costing further performance.

2.1.2 Asymmetric Redundancy

A typical way to reduce the resource usage of the replicated safety-relevant software from the previous approach is to not replicate it. Instead of executing the safety-relevant controlling software twice (once on each core), it is executed only once and a separate checking software is executed that only checks whether the controlling software is operating correctly. The checking software does not control the actuators itself. This is typically known as the master checker approach and reduces the processor load since it is usually simpler to check than to control. In the example shown in Figure 2, the safety-relevant controlling software is executed on the first core and the checker software is executed on the second core. The checker software is decomposed into the actual check and the software calculating acceptable master results.

Figure 2 Master Checker Setup



This example contains a disable signal that the checker issues to signal an error condition to the system if a check fails. Other actuator control approaches, as shown in section 2.2, can also be deployed but it is important to note that the core executing the checker software typically is not able to generate control signals since it does not execute all the necessary operations to generate such signals.

It is also possible to implement an overall model of the system as a checker and use the model to check whether the commands issued by the master were actually performed by the actuators (represented by the dotted line from the processing stage to the check target generation). In addition, such system models can be used to assess whether the values of different sensors correlate sufficiently or develop consistently over time. The details of such an approach are highly application specific and outside of the scope of this paper.

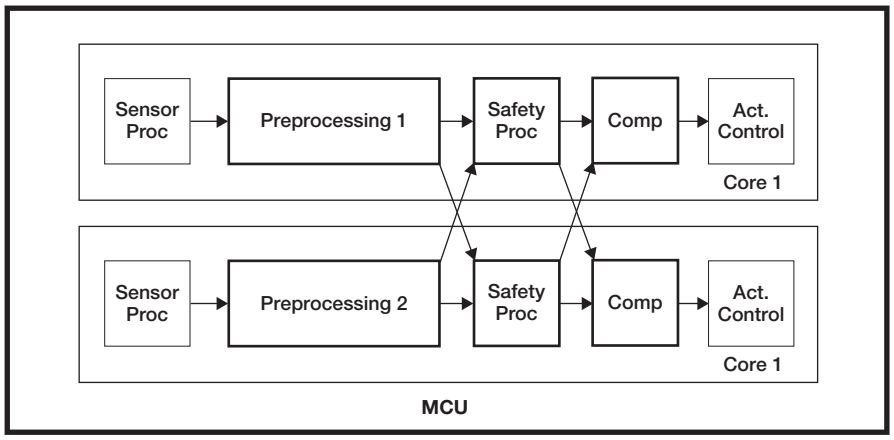
2.1.3 Partial Redundancy

Another approach to address functional safety requirements in DPM involves replicating only parts of the overall software included in the safety function. This approach is feasible if an appropriate split into preprocessing and safety processing tasks exists.

In the first step, different functions are executed on both cores; respectively the same function is executed on different data (called “Preprocessing n” in Figure 3). The results of this first, non-replicated step are distributed between the cores. Using the preprocessed information, the safety-relevant aspects of the overall function are executed on both cores (“Safety Proc.” in the figure). The idea is that these latter functions not only further process the results but can also detect or handle all failures originating from the preprocessing steps. Failures in the safety processing functions are detected by their replication and the subsequent comparison step similar to the architecture in section 2.1.1.

[1] If RAM is error correction code (ECC)-protected, which is the typical case nowadays, duplicated storage is not necessary for achieving sufficient error detection.

Figure 3 Partially Split Processing

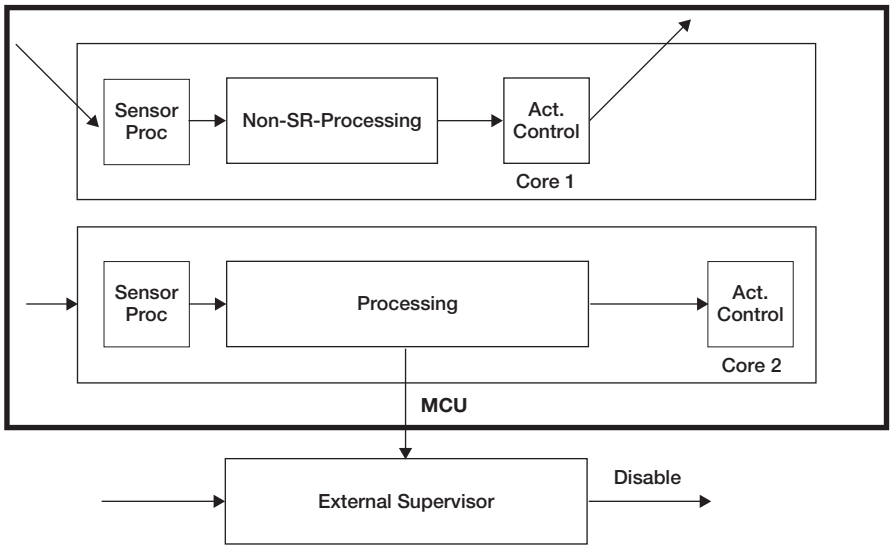


Allocating functions in the way given by this software architecture reduces the MCU load since the preprocessing steps are not replicated. If a significant amount of work can be put into those steps, the performance of the system can approach a non-safety-relevant dual-core system.

2.1.4 External Redundancy

Another way to implement a dual-core MCU in DPM does not use the second core for safety-relevant tasks at all. In this approach, all of the safety-related functions are not placed on the MCU. Instead, an external supervisor (see Figure 4 below) checks the correct execution of the safety-relevant processing steps that are executed only on one core. If the supervisor's calculations disagree with the result from the processing step, it will bring the system into a safe state by disabling the actuators.

Figure 4 External Safety Measures



Because the safety-relevant (SR) functions are executed on one core only, the second core can be used for other processing. This approach is viable if one does not want to develop the non-safety-relevant-processing functions according to IEC61508 or ISO26262. If this is not the case, one has to demonstrate freedom of interference[1] between safety-relevant and non-safety-relevant functions. The given architecture supports freedom of interference due to the separation of the cores with individual crossbar, RAM controller and other local modules. It is an interesting architecture if an existing safety concept (including the external supervisor) is reused and/or also integrated with other functions.

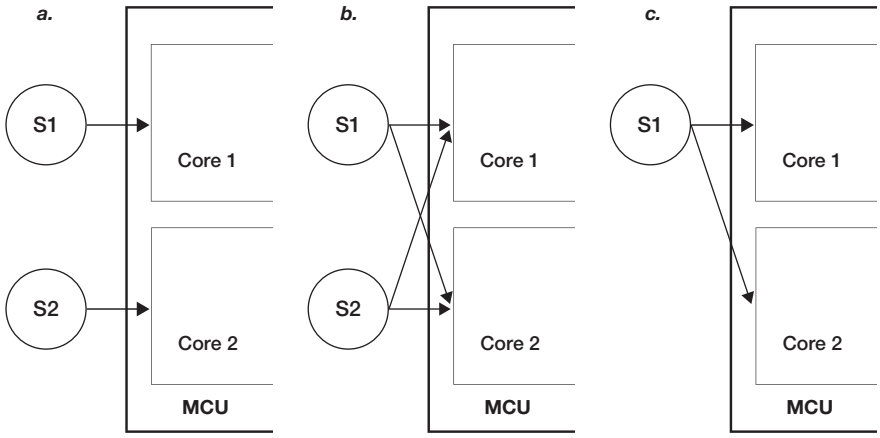
Overall, using the DPM provides a higher amount of flexibility regarding how and where safety strategies are applied, but with the secondary effect of varying degrees of increased system complexity, depending on the additional software needed.

2.2 Sensor Input

Acquiring and processing sensor inputs in safety-relevant systems is typically very application specific since it is possible to choose from a wide variety of options[2]. This variety increases in DPM due to the different approaches for computation.

Sensor data can be propagated in multiple ways to the computational parts of the software, some of which are depicted in Figure 5.

Figure 5



a) Two-channel sensors; b) fully connected, redundant sensors; c) single self-checking sensor

Figure 5a shows an approach with two similar sensors connected in a two channel configuration, where each sensor is connected to one core. This is a likely method to connect sensors in architectures such as those from Section 2.1.2 as well as Section 2.1.4 (and to core 2 and the external supervisor). Due to inputs from two different sensors, equality comparisons in the test stage are not possible. By using physical principles correlating the sensor values, a sensible check can be designed.

In Figure 5b, all sensors forward their data to both cores allowing the computational parts to generate equal results. This requires special measures that are described below. Typically, in this approach, the sensor values will be explicitly compared and acceptance tested when entering each redundant input, whereas in the previous two-channel connection, they are implicitly compared by the final comparison stage.

Figure 5c shows the case of a single sensor connected to both cores. If the sensor is safety relevant, it will probably be self checking and protecting its transmissions with a special communication protocol (see below). In this approach both cores can receive equal sensor data, allowing equality comparisons in the checking stage.

[1] See ISO 26262-6 Annex D
[2] This can be seen from the Safety Application Guide of MPC564xL [1] that lists several ways for input acquisition but is not complete.

Figure 6 SW Components for Sensor Processing

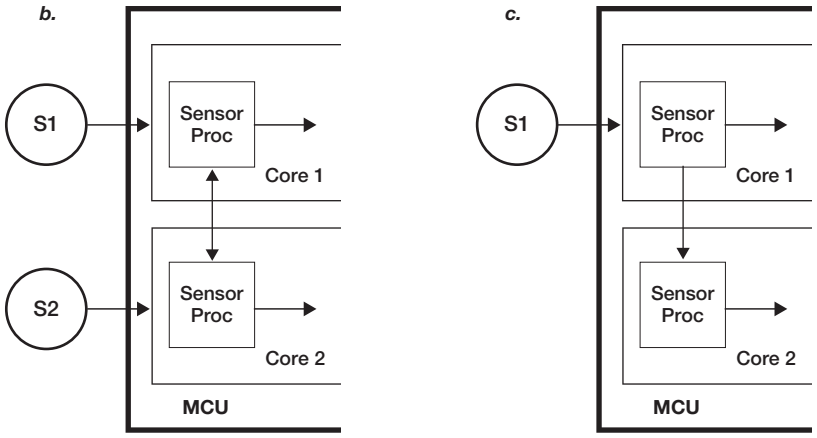


Figure 5 presented a logical view on the data flow between sensors and the software executing on the cores. Actual implementation might look different as demonstrated in Figure 6, which includes possible setups of sensor processing components that were ghosted in the figures of Section 2.1.

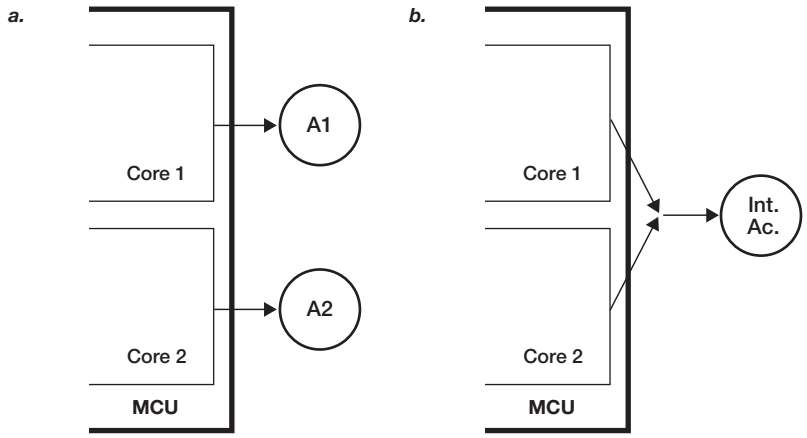
In the fully connected approach from Figure 5b, actual wiring should connect each sensor only once. Exchange of results between the cores is handled by software that is also responsible for synchronizing sensor accesses. Without such synchronization, retrieved sensor values can vary significantly due to the different retrieval times. Depending on the application, this can impede the determination of a maximally allowed divergence of results for the comparison stage. In general, this is known as the problem of replica determinism (see [4]).

In the approach of a single self-checking sensor shown in Figure 5c, there is typically only one physical connection from the sensor to the dual-core MCU. Reading the sensor twice via this single connection (once by each core) leads to the problem of replica determinism. Alternatively, the approach shown in Figure 6 can be followed, where only one core actually reads data from the sensor and forwards it to the other core. Obviously, this core is now a single point of failure. However, the communication protocol the self-checking sensor typically runs can be used to prevent this single point problem if the protocol includes data protection. The communication protocol (or at least the data protection part of it) has to be executed independently by both cores locally so both cores can verify that the received sensor data is genuine.

2.3 Actuator Access

Similar to sensors, actuators can be connected in different ways to an MCU in DPM. Some will be listed here and examined in the next section for additional failure modes when used in DPM.

Figure 7 Direct Actuator Connection

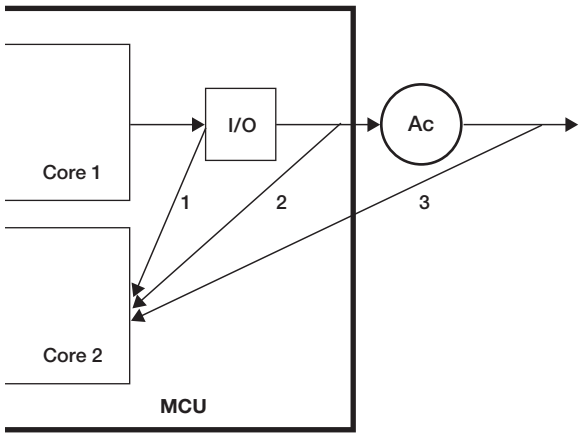


a) Two-channel actuator control, b) Merged, single actuator control

The connection methodology shown in Figure 7a is very similar to the Dual Write approaches given in MPC564xL's Safety Application Guide [3], in section 2.10.4.2. In this example, two actuators, A1 and A2, can independently control the behavior of the system to avoid a dangerous state (e.g., the high-side switch and the low-side switch of an airbag squib) or one of the two outputs can be a Disable line that shuts off the actuator, controlled by the other line, if the respective core detects an error.

Figure 7b shows how to control a single actuator without a separate disable line. In this case, a single failing core is prevented from sending dangerously wrong commands to the actuator. To achieve this, a technique must be found (depending on the type of communication) to merge contributions of both cores in such a way that one alone cannot activate the actuator. An example of this would be the control of an intelligent actuator attached via CAN. One core would write the payload, the other calculate the application checksum and add it to the data already written by the first core. If the first core detects an error, it writes no payload. If the second core detects one, it writes no CRC. In both cases, no valid command is sent to the actuator.

Figure 8 Actuator Feedback Supervision



The last actuator example, shown in Figure 8, uses feedback mechanisms. It can partially be found in MPC564xL's Safety Application Guide [3] in sections 2.10.4.1 and 2.10.5.2 as with readback methodologies. Feedback point 1 corresponds to the Internal Readback Configuration, point 2 to the External Feedback Configuration and point 3 is not mentioned. Depending on the point the feedback is taken from, a different set of failures can be detected. Also, the effort to gather the feedback differs: Point 1 requires a read from an MCU register, point 2 needs an additional input into the MCU and reading data from point 3 typically requires sensors and a way to calculate the expected behavior of the system given the actuator commands of Core 1. Such an approach will likely be similar to the system model approaches mentioned in section 2.1.2.

2.4 Failure Modes of Input and Output Operations

In principle, functional safety for sensor and actuator access does not pose a new problem. It is also a topic in a system setup with LSM or with no replication at all. Yet, when compared with an MCU in LSM, there are some differences to note in how input and output operations can fail. This section will focus on failure modes possible in DPM on the MPC564xL.

2.4.1 Single Points of Failure for Sensor Accesses

In general, operating an MCU in DPM instead of LSM typically increases the number of single failure points even when sensor results are accessed from two different cores. A permanent fault in such a single point of failure can change all redundant data passing through the point in the same way, remain undetected and thereby propagate its failure. A transient fault should normally influence only one sensor value and thus be detectable or even acceptable. In MPC564xL MCUs for example, additional single points of failure exist in DPM that will be explained in this section.

2.4.1.1 DMA

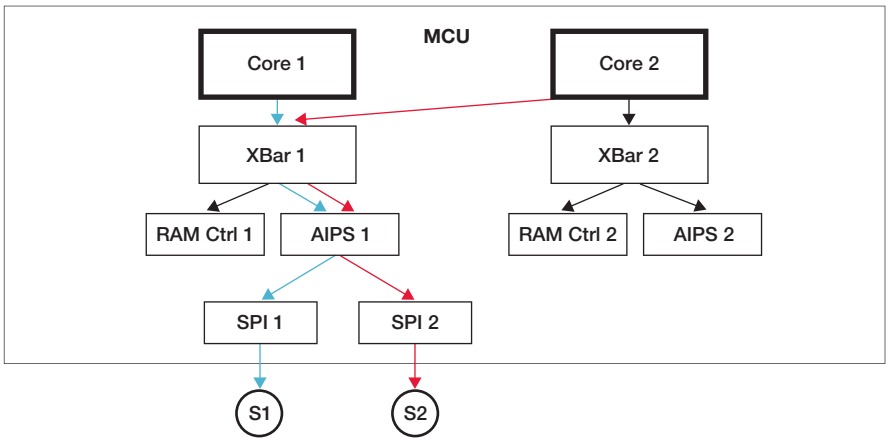
In DPM, only one of the MPC564xL's two DMA engines executes. Any data moved by DMA is left open for damage by the DMA engine. If two sensors are used (such as the configurations shown in Figure 5a and 5b), at most, one of them should be fetched from the I/O registers with DMA. Otherwise, a permanent error in the DMA could corrupt both sensor values in the same way and prevent detection by comparison.

The problem does not exist if the two sensors fetched by DMA are diverse and a lack of DMA execution can be detected. Both values could be corrupted by a single error in the DMA but the semantic effect of that corruption would be different and likely be detected.

2.4.1.2 I/O Bridge

The I/O bridge converts transfers from the MCUs main bus to the peripheral bus and vice versa. In the MPC564xL, the I/O bridge is called AIPS and the MCU has two AIPS blocks. In DPM, all I/O components are connected to only one of them. A permanent failure in that AIPS can thus corrupt data coming from several sensors even if those are connected to different I/O modules as can be seen in Figure 9 for two SPI sensors.

Figure 9 Single Point of Failure in Data Path



In LSM, such an error would be detected due to the replicated operation of the AIPS. In DP-mode, this coupling does not exist.

2.4.1.3 Bus/Crossbar

All transfers from I/O subcomponents to the cores normally pass through a bus. In MPC564xL MCUs, these are two crossbars (see XBar in Figure 9 above). If all I/O components are connected to the same bus, as they are in the MPC564xL in DPM, the bus becomes a single point of failure. The failure modes are the same as the I/O bridge.

2.4.1.4 SRAM Controller

MPC564xL MCUs have two controllers for their system RAM. The RAM array is split in half and each half is managed by its own controller that is connected to one of the two crossbars (see Figure 9 above). The arrays are protected by an ECC calculated by the controller over data and address buses and are not a single point of failure. This is not true for those parts of the controller that are not protected by the ECC (including the ECC logic itself).

If data for redundant processing was stored in the same half of the RAM, the corresponding RAM controller constitutes a single point of failure since it can corrupt the data going to both redundant calculations. This can also happen if data is stored in one “half” of the RAM and then copied to the other half to create a redundant copy. To avoid this, redundant copies should be created by directly reading the sensors.

2.4.1.5 I/O Module

Not surprisingly (and similar to the LSM case), an I/O component can be a single point of failure if two redundant sensors are connected to the same component. This is the reason for duplicating most of the I/O modules of MPC564xL MCUs. This redundancy should be used by the application.

2.4.2 Single Points of Failure for Actuator Control

For actuator output, the DPM adds the same single-failure points compared to LSM as it does for sensor input (see section 2.4.1). These impact the Dual Write setup (Figure 7a) the most because they could influence the data sent on both channels in the same manner. Also, feedback point 1 of the Feedback approach (Figure 8) is at risk from some less likely errors if the failure causes the error to be masked during readback. Feedback points 2 and 3 are likely to be less endangered as they include an additional sensor in their path injecting diversity since the sensor feedback results are different from actuator control commands.

3 Special Measures for DPM

In MPC564xL MCUs, DPM disables the automatic hardware comparison of cores, caches, memory management units, crossbars, memory protection units, RAM controllers, AIPS, interrupt controllers, software watchdogs, software timers, MCM and DMA (see [2]). If these components are used in a safety-relevant function, they must therefore be checked, either by the software itself or the software must enable additional checkers included in the hardware. Some proposals on how to achieve this will be given here.

Since neither of the two cores can be trusted individually, safety-relevant software has to be executed on them redundantly. Some possible solutions were shown in section 2.1. According to those software architectures, redundant execution of safety-relevant software on both channels takes care of faults in core, cache and MMU. For other components, measures will be proposed in the following sections.

3.1 Prevention of Channel Interference

Preventing interference between the two channels on the MCU is necessary to enable the two channels to check the correctness of each other. Otherwise, error propagation could lead to dependent failures in both channels making the failure undetectable. To achieve this, the following measures are recommended.

On-platform periphery modules closely attached to a core such as interrupt controller or watchdog, should only be used by its respective channel. This matched usage is especially relevant for SWT and STM. A process running on core 1 should trigger only SWT 1 and retrieve any timing information only from its local system timer module. To ensure matched usage, access to on-platform periphery of the channel should be restricted with MMU, MPU or AIPS. Also, matched usage must be taken into account when a multi-core operating system (OS) is used. Often, such an OS has most of its functions on one fixed core and uses only the local timer but manages tasks on both cores. This makes the timer a single point of failure for the task control of both cores. It could be detected by checking one core's timer with the other core's timer.

System RAM should similarly be used mainly in-channel. Each channel includes a RAM-controller and half of the overall RAM array. Stacks, temporary results and similar data should be located appropriately in the respective half. Some input data might require placement in both RAM halves. For example, the sensor architecture shown in Figure 6c should copy the input data into both RAM halves before checking the CRC. Also, data retrieved by double reads from sensors should be placed in different RAM halves. Note that the RAM halves have different addresses, so software executing replicated over both cores from a single code copy must either employ relative addressing or use the MMU.

Data that is used by both channels but placed in RAM only once (e.g., to allow information exchange between both cores) should be protected by an application CRC or similar measures to help detect failures of the respective RAM controller.

A special case is the ***HW semaphore*** implementation, which allows coordinated access of both cores to concurrently used resources and avoids race conditions. This is an important service in DPM mode[2] but, by their nature, the semaphores can break the isolation of the two channels. A non-working semaphore on one core can either block both cores (by incorrectly telling both cores that the resource is locked by the other) or not block any core (by incorrectly telling both cores that the resource is not locked by the other) causing race conditions. The first condition will be detected by the SWT (see Chapter 4). The latter situation needs to be detected by software, possibly with an overlaid software semaphore implementation.

Since both cores have access to both ***AMBA crossbars,*** there is the possibility that one core could block or significantly delay the other core's accesses from passing through the crossbar. This risk should be reduced by appropriately assigned arbitration priorities at the crossbar clients (e.g., by giving each core priority within its own channel). Unfortunately, it is not completely possible to prevent this failure mode since that would require restricting all interaction between the cores and some kind of interaction between both cores is necessary for most of the architectures discussed in Section 2.1. The remaining failures will have to be detected by the SWTs.

The address isolation measures of ***MMU, MPU and AIPS,*** which were already mentioned above, can restrict access to RAM areas or I/O registers based on the requesting bus master or even the requestor's process ID. However, the address isolation measures need to be configured in such a way that they ensure isolation between the two channels as well as isolation between software entities of different safety integrity levels running in the same channel. For example, in the architecture given in section 2.1.3, one probably wants to isolate

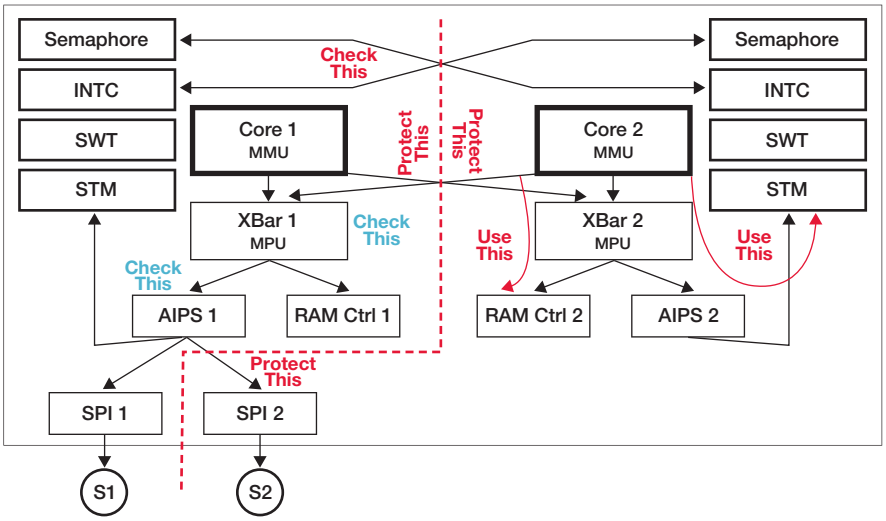
[1] Note that on MPC564xL MCUs, even in DPM mode, the flash-controllers will run in lockstep mode
[2] See Chapter 7.6 in [2] for an application example

the pre-processing software from the post-processing and checking software to allow them to be developed according to different Safety Integrity Level (SIL) ratings. If different processes have different access rights, this could mean that MMU, MPU and/or AIPS configuration need to be changed on each process switch if the available address regions are insufficient.

One important point to remember is that the configuration registers of MPU and AIPS also need to be protected against interference from the other channel so that the protection granted by MPU or AIPS cannot be disabled by that channel.

Figure 10 shows the previously discussed interference protection measures (in red) together with the checking measure for AMBA crossbar and AIPS peripheral bridge from the next section (in blue).

Figure 10 Interference Protection Measures



3.2 Single Point of Failure Supervision

Section 2.4.1 has shown that the deactivation of redundant hardware execution in DPM introduces several single points of failure where a single, permanent fault can become dangerous due to propagation to both channels. This section lists possible counter measures against such failures.

Special measures might be necessary for the ***INTC.*** An interrupt service routine will typically only be executed by one of the cores. If the results are used by both channels, failures of the respective INTC or the whole execution channel can spread to the other channel. Some software architectures can check for such problems due to sensor diversity, self-checking sensors or indirect checking (section 2.1.3). In other cases, it may be necessary to directly supervise the results of an interrupt handler and/or the correct triggering of the interrupt by the INTC using the second core. An alternative could be to periodically switch IRQ handling between the two channels during the system safety time by dynamically reprogramming the INTC.

A similar situation exists with the ***DMA.*** It is slightly more complex because there is no active second DMA engine that could be used for software-triggered supervision of the results of the first one. For this reason, it is advisable not to use DMA for safety-relevant tasks, unless other checks are available to detect DMA failure effects (e.g., diverse sensors, second sensor read directly by core, etc.). If DMA usage is unavoidable, some software checks, preferably executed by the second core, need to be included to prevent wrongly copied data influencing the safety-relevant functions executing on both cores:

- A simple measure could be canary words at the start and end of the DMA target area (outside and inside of the area) to check that the DMA did write exactly the expected number of words. This does not prevent data corruption by the DMA and is not workable if the targets are I/O registers.

- Alternatively, an online self-test of the DMA engine can be conducted. But sufficient coverage within the system safety time might be difficult to achieve due to the inability to check the same address, the same internal DMA engine descriptor, the same IRQ and other comparisons during the functional operation of the DMA.

Even if the DMA is used only for non-safety-relevant purposes, access of the DMA engine to crossbar clients should be strictly restricted using the MPU and AIPS to avoid broken DMA interference with safety-related functions.

The ***AIPS*** I/O bridges cannot be used to check each other in DPM. As described above, all off-platform peripheral modules[1] are connected to the first AIPS module with the second remaining idle except for connections to its core-attached peripheral functions. The AIPS is therefore the most prominent single point of failure for redundant sensor or actuator accesses. A possible countermeasure is reading the I/O modules' configuration registers. For LSM, the safety concept requires that configuration registers are read and compared with the expected values (e.g., by calculating a CRC over them) at least once per system safety time to detect bit flips in them. In DPM, this configuration scan is still necessary. In addition, read and compare is highly recommended to ensure that it:

- Includes all safety-relevant I/O modules (to ensure they can actually be addressed)
- Reads registers that contain values so each data bit is set and unset at least once when passing the AIPS
- Reads registers from addresses so each relevant address bit (i.e., used to address safety relevant registers in the I/Os) is set and unset at least once.
- Executes some write operations and checks their success

To ensure this, it may be necessary to read data from non-configuration registers.

There is another reason to read I/O configuration registers regularly. A transient addressing error during data writing might overwrite a register causing a misconfiguration of an input path or even wrong data to be sent along an output path.

The first ***AMBA*** crossbar is a single point of failure for I/O accesses similar to the first AIPS, since both cores use parts of the crossbar to reach the AIPS (see Figure 10). The online self test for the AIPS defined above should take care of possible single point failures on the crossbar as well.

3.3 Other Recommendations

The clock monitoring of MPC564xL MCUs only supervises the clock of core 1 but not of core 2. Therefore, failures in clock management (but not clock generation) could lead to the second core experiencing clock failures without detection by some hardware mechanism. It is recommended that at least some safety-relevant functions are executed on the first core and that either these functions or the SWT of the second core supervises the functions implemented on the second core.

4 LS Mode Measures Also Relevant for DP Mode

MPC564xL's Safety Application Guide ((SAG) [3]) mentions several measures that need to be executed by the user to ensure SIL 3 capability in LSM. In general, these measures are also useful in DPM. For this reason, several of the measures mentioned in Chapters 2.2 and 2.3 of this document are also listed in Chapter 2.10 of the SAG.

In practice, it is more difficult to define which LSM measure also needs to be executed in DPM. This depends significantly on the actual application and the software architectures chosen. Nevertheless, a few of the LSM measures will be presented since they normally provide protection against faults also in DPM.

Several points already mentioned the ***software watchdogs*** (SWT) as primary or fall-back failure detection measures. This is the same as in LSM, where the SWT is implemented to detect software faults (crashes, loops, etc.) as well as some hardware errors.

In DPM, the SWT can also provide an indication if a single core is executing improperly or if it is stuck. It is therefore a first line of defense against failures of a single core which could prevent execution of safety functions or (try to) overwrite the data of the other channel. The SWTs should therefore be used within all safety-relevant processes so the blocking of a single one can be detected.

[1] This applies specifically to peripheral modules that are not closely attached to a core.

Note that MPC564xL's LSM safety concept requires an additional simple ***external watchdog*** to be triggered by software. This is the same in DPM since this external watchdog is intended to detect common-cause failures overlooked by the safety analysis.

Section 2.10 (I/O Peripherals) of the SAG mentions the need to periodically verify the content of several ***configuration registers*** within system safety time. This need continues to exist in DPM (see also section 3.2 above) but now also concerns configuration registers of components that were protected by replication in LSM. Therefore, the configuration registers of XBar0 and AIPS0 should be periodically read as well as both channel's MCM registers. The latter is necessary to control the ECC logic and various other components included in the MCM registers. Other configuration registers may also need periodic verification depending on the actual application.

Section 2.6 (flash memory) of the SAG requires periodically testing the ***ECC logic*** of the flash. This is necessary because the flash's ECC logic is within the flash array and therefore not replicated with the flash controllers. In DPM, additional testing of the ECC logic in one or both SRAM controllers may be necessary depending on how the safety-relevant applications makes use of the system RAM. Note that if safety-relevant data is always replicated in both RAM halves, no RAM ECC logic tests are necessary.

Similar to LSM, the ***FlexRay™*** controller is a master on the AMBA crossbar but is not supervised against failures. Therefore, special care must be taken to prevent it from corrupting safety-relevant data of both channels. Typically, this will be done with the MPU by restricting its access to those memory areas containing its buffer structures.

5 Faults Handled by Safety Measures within MPC564xL MCUs

The previous chapters described what the MCU user must do to achieve sufficient safety integrity in DPM. MPC564xL MCUs also provide measures to detect and handle failures in that mode. A short overview of these measures provides insight to the failures that do not need software detection measures.

All safety measures from LSM that do not rely on component replication are available in DPM. This includes the startup self test, the common-cause failure countermeasures and the single-point failure countermeasures based on information redundancy.

5.1 Latent Faults

MPC564xL's **self tests** ensure correctness of the MCU at startup time. The self tests detect permanent, latent faults in the digital logic as well as the memory. Also, some of the analog I/O operations, such as the ADCs, can be self-tested. Diagnostic coverage numbers can be found in [2]. The logic and memory self test execute automatically during start up, but the peripheral self tests have to be triggered, primarily during run-time as described in the SAG.

5.2 Common Cause Failures

MPC564xL MCUs contain measures against common-cause failures that can occur from the usage of a **common die** for the two execution channels. These measures are required by IEC 61508-2 Version 2, appendix E for any device using redundancy on a single die. The measures are therefore also required for DPM with software replication. They concern the placement of modules and routing of signal lines on the chip and require no software interaction to become effective.

In addition, the correct values of several **control signals** are supervised to ensure that the MCU does not switch into a mode where safety measures might be disabled or normal functionality might be broken.

Faults in the **clock generation** or **power management** circuitry can influence the execution of software on both cores. In principle, the probability of both channels being influenced in the same way by a clock or power failure is quite low and can be further reduced by executing the replicated software at different times. Nevertheless, MPC564xL MCUs contain measures to detect over- or undervoltage as well as clock loss and frequency deviations. The FCCU (see 5.3) should be programmed to react appropriately based on the reports of such failures.

5.3 Single Point Faults

The MPC564xL does not replicate its **RAM** and **flash** to protect it against bit corruption and addressing errors. Instead, it follows an information redundancy concept using a SEC/DED ECC code for both RAM and flash. Due to additional measures, such as interleaving, the ECC detects random bit flips caused by to alpha and neutron radiation with a diagnostic coverage up to 99 percent. This is still true in DPM. Yet, as mentioned in Chapter 4, in DPM, the task of ensuring the correctness of ECC operation could fall on the software.

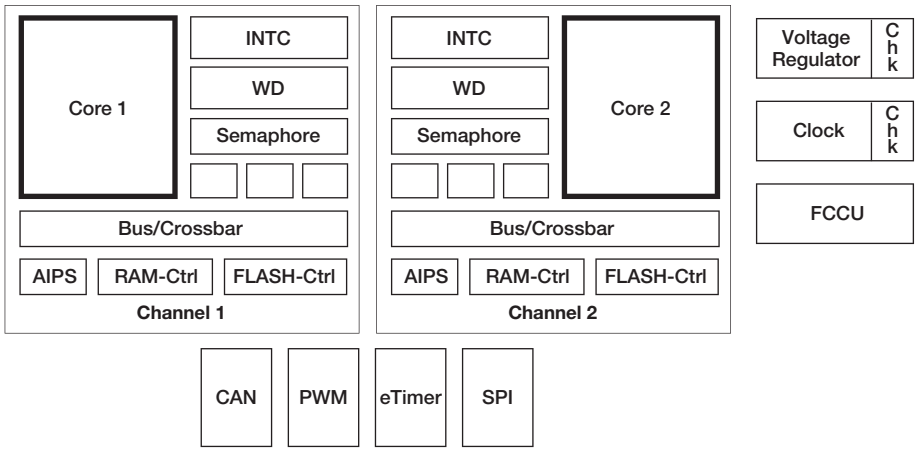
Finally, MPC564xL MCUs contain a separate unit (the **FCCU**) to collect **failure reports** from the different subcomponents of the MCU and determine an appropriate reaction (based on user configuration). The FCCU is essentially independent of the rest of the MCU. This enables it to process failure information even if other parts of the MCU are faulty. The FCCU continues to work in DPM and is triggered by failures of the clock or power regulator. It can also be used by the software to signal failures via the error out pins (F[0:1]) to trigger external fault reactions.

All of these measures are effective in DPM, even though they were designed originally for Lockstep mode. The dual use of safety measures simplifies development of a safety concept for the DPM.

6 Differences to Lockstep Mode

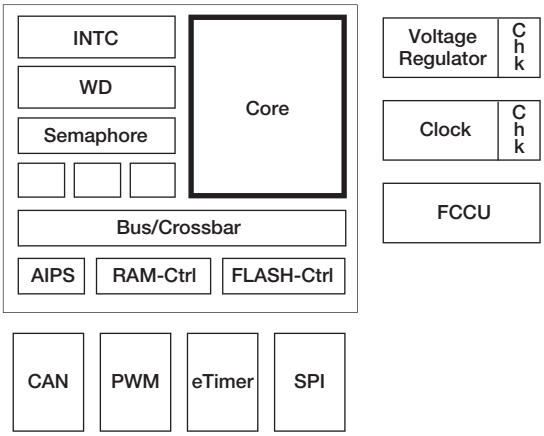
As mentioned in Chapter 3, MPC564xL's lockstep mode uses redundancy as a safety measure for several of its components. As a result, the DPM safety measures discussed in Chapter 4, such as the in-channel usage of components and RAM, the strict software supervision necessary for DMA and INTC, or the regular self tests of correct addressing by the peripheral bridges, are not necessary in LSM. Failures in the corresponding modules will be detected by hardware when the two channels behave differently.

Figure 11 SW View of DP Mode



However, the overall software architecture differs between DPM and LSM. Figure 11 shows in an abstract way, how software sees the components of Leopard in DPM. Section 2.1 presented several architectures for DPM where two software entities were running distributed over the two cores that checked each other's execution. This requires additional specification work for defining comparison concepts and thresholds as well as implementation work for communication and synchronization between the software entities, especially for sensor and actuator accesses. Also, both software entities will typically require local RAM storage for intermediate results and to maintain state information between calculation cycles. The software measures mentioned in the previous chapters are at least partially dependent on the specific hardware in use and as a result, potentially restrict portability of the code.

Figure 12 SW View of LS Mode



In LSM software sees only one core, as shown in Figure 12. Hardware hides the fact that the software is actually executed on two cores. Therefore, there is no need to synchronize software over two cores. Also, the comparison between the two cores is executed in hardware, so no explicit comparator software must be written. Finally, since our lockstep safety concept does not require replication of variables, jump tables, etc. in RAM, the RAM usage is lower than in DPM.

Overall, LSM reduces and simplifies user-written software and reduces RAM requirements. LSM also comes with a predefined safety concept and instructions on how to include it into the overall system. For DPM, this paper only touches on possible safety concepts due to the large variety of potential configurations that could be employed. Ultimately, the selection of the appropriate concepts depends on the application safety strategy employed by the user. On the other hand, DPM can more efficiently use the available performance of the MPC564xL because it executes non-safety-related software only on one core whereas this code would be executed automatically on both cores in LSM. Finally, the DPM use is most appropriate with a diverse software implementation.

The decision between lockstep mode and decoupled parallel mode depends on the application, intended safety concept and the required performance, as well as the acceptable software effort.

Bibliography

[1] Avizienis, A., “The Methodology of N-Version Programming,” *Software fault tolerance,* pp. 23-46, Lyu M. (Ed.), Wiley, 1995.

[2] *Leopard Reference Manual,* Rev. 3, Freescale Semiconductor Inc., Austin, TX, 2009.

[3] *Safety Application Guide for Leopard,* Rev. 1.6, Freescale Semiconductor Inc., Austin, TX, 2009.

[4] Poledna, S., et al, “Replica Determinism and Flexible Scheduling in Hard Real-Time Dependable Systems,” *IEEE Transactions on Computer* (2000) 49: pp. 100-111.

[5] Siewiorek, D. and Swarz, R., *Reliable Computer Systems,* 3rd edition, A.K. Peters, Natick, MA, 1998

How to Reach Us:

Home Page:
www.freescale.com

Power Architecture Information:
www.freescale.com/powerarchitecture

Web Support:
www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate,
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447
303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: MPC564XLWP
REV 0

freescale semiconductor