AN4417
Application note

SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx devices Exception handling and single/double bit error

Introduction

This document provides an overview of SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx exception handling with main focus on different kinds of exception that the application code may face during the runtime like single and double bit errors in memories, MPU protection violation, AIPS access protection violation and others.

It starts with the simple overview of Machine check interrupt highlighting important things from an application perspective. To get detailed view and to implement low level machine check interrupt handler, it is necessary to use Z4 Core User Manual which describes all the details about the Core exception and interrupts.

The following part describes the reason of the exception, how to find it and what possibilities exist to remove the fault.

Contents

List of tables

List of figures

1 Z4 Core exception overview

Z4 Core used on SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx devices contains many exception sources and sixteen interrupts to service them. Multiple exception sources can be mapped to one interrupt handler where few supportive status registers provide flags to find the cause of the exception in the handler.

A detailed list of the exception causes and their mapping to interrupt handlers is found in the Z4 Core Reference Manual (see *Section Appendix B: Reference documents*).

This chapter gives an overview of machine check interrupt that is utilized for several important fault states of SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx devices.

1.1 Machine check interrupt (IVOR1)

Machine check interrupt is a handler that services multiple fault events that may occur during runtime code execution.

Table 1. Machine check interrupt causes

Interrupt type	Exception conditions
Machine check	– NMI – ISI, ITLB, Error on first instruction fetch for an exception handler – Parity Error signaled on cache access – External bus error

This interrupt is used to handle various faults generated by peripherals in the SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx devices, like MPU protection fault, 2b ECC error in the Flash or RAM memory etc. The reason is that most of the faults are signaled back as an external bus error situation during the CPU-Submodule bus transaction.

1.1.1 Machine check registers

Z4 core implements few machine check status registers that are updated upon the exception event with some constraints stated in the Z4 Core Reference Manual (see *Section Appendix B: Reference documents*). These registers are used to find the source of the exception and based on it, it is decided how to solve it.

Table 2. Machine check register

Register	Content
MCSR (syndrome register)	Register indicates the source of machine check, this condition gives the possibility to differentiate between them
MCAR (address capture)	Register contains some sort of machine check conditions, the address for which the asynchronous type of the machine check exception was raised. Address valid only when MCSR.MAV bit was '0' before the exception, otherwise MCAR register is not updated.
MCSRR0 (Save/Restore register)	Address of the instruction that caused the exception. Once the exception is finished (mcrfi instruction), program starts execution with the same instruction, that was the cause of the exception.

Machine check syndrome register (MCSR)

This register is the first register to check additional information about the cause of the exception. There are three groups of machine check causes.

Table 3. Machine check causes

Machine check cause	Brief description
Error Report Machine check (IF,LD,ST,G)	These exceptions are directly associated with the current instruction execution stream. They are not masked with MSR_{ME} bit. It means that the exception is always taken whenever the condition occurs. They differentiate among Instruction fetch, Data store and load.
Non-maskable interrupt (NMI)	Not MSR_{ME} gated exception occurs when NMI signaling is enabled and NMI pin is driven low.
Asynchronous Machine check (BUS_IREER, BUS_DRERR, BUS_WRERR)	Exceptions reported by the subsystem, usually as bus error termination, back to the Core. They are enabled by MSR_{ME} bit and are cumulative. This machine check whether the exception group triggers capture of the corresponding address to the MCAR register and if $MCSR_{MAV}$ bit is cleared. If $MCSR_{MAV}$ was previously set, then the MCAR register is not affected.

Machine check address register (MCAR)

MCAR register contains target address reporting the fault condition. It is updated only for Asynchronous Machine check group when MCSR.MAV bit is cleared and it is valid only if MCSR.MAV status flag is set. Otherwise the MCAR register cannot be used in the fault analysis.

It is important to clear MCSR.MAV bit after reading MCAR register value to enable the capture of the address, in case of new asynchronous machine check fault.

Machine check MCSRR0 register

This register is updated by the HW at the beginning of the machine check interrupt. It stores the address of the instruction that causes the error condition.

It is used at the end of the machine check when *mcrfi* instruction is executed to fill the instruction pointer. The result is that code restarts the same instruction that was the cause of the error, if additional modification of the MCSRR0 register is not explicitly done.

2 Machine check handler

Machine check handler usually splits into two parts:

- Low level handler
- User handler

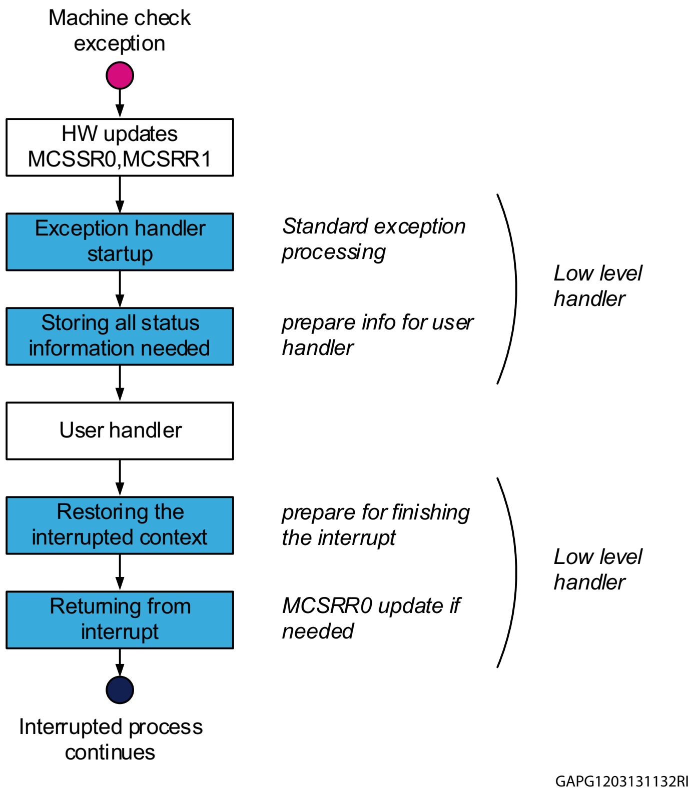
2.1 Low level handler

Low level handler is responsible for the first and the last part of the exception execution. It is usually written in assembly language as it needs to execute the proper instruction sequence before it can pass the code execution the higher level routine and accesses special purpose Core registers.

The middle of the interrupt service routine belongs to the user handler where analysis of the root cause of the exception and fault removal is done.

Once the user handler has finished, code execution is given back to the low level driver to finish the interrupt and return back to the interrupt process.

Figure 1. Machine check exception flow


Machine check exception
HW updates MCSSR0,MCSRR1
Exception handler startup
Storing all status information needed
User handler
Restoring the interrupted context
Returning from interrupt
Interrupted process continues
Standard exception processing
prepare info for user handler
Low level handler
prepare for finishing the interrupt
MCSRR0 update if needed
Low level handler
GAPG1203131132RI

2.1.1 Start phase

In the first step the hardware (Core) carries out checks, when the machine occurs exception. The hardware stores content of the MSR register and address of the current instruction pointer if it is possible (precise exception), immediately low level driver starts processing.The machine executes several steps like check status register saving, context of the interrupted process saving and others. This part stores some additional information as they are used by the higher layer user handler to analyze the root cause of the exception later.

In the Z4 Core User Manual documentation there is a detailed description of the machine check resources, their meaning and proper handling in case of interrupt. Low level handler follows rules and recommendation described in the User Manual (see *Section Appendix B: Reference documents*).

2.1.2 Final phase

Here the handler should restore the saved context of the interrupted process and return with the *mcrfi* instruction.

Before *mcrfi* instruction is executed, which fills instruction pointer with MCSRR0 content and MSR register with MCSRR1 content, MCSRR0 modification might be needed.

There are two cases which determine if the manipulation is needed or not. This information is useful in the user handler to pass down to the low level driver.

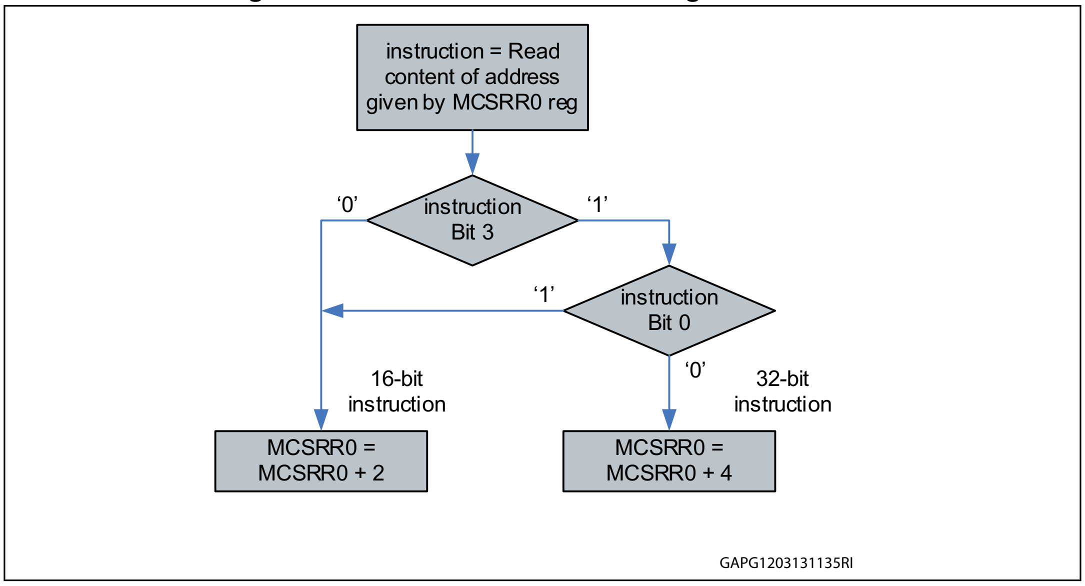
1. User handler finds the cause of the machine check exception and fix it in a way that the program can re-execute the same instruction that caused the machine check exception.
2. User is able to find the cause of the exception, but the problem remains and re-executing the same instruction lead to the machine check exception again => Modification of the MCSRR0 is needed.

2.1.3 Modification of the MCSRR0 register

In case the exception cause of the exception cannot be removed, MCSRR0 register value is modified in a way that it takes the address of the following instruction. This prevents re-execution of the faulty instruction and retriggering the machine check exception.

Modification has to consider VLE instruction coding in case the interrupted process is implemented in VLE coding and the value increased according to the length of the faulty instruction pointed by the current MCSRR0 register content, see *Figure 2*.

Figure 2. Modification of MCSRR0 register content



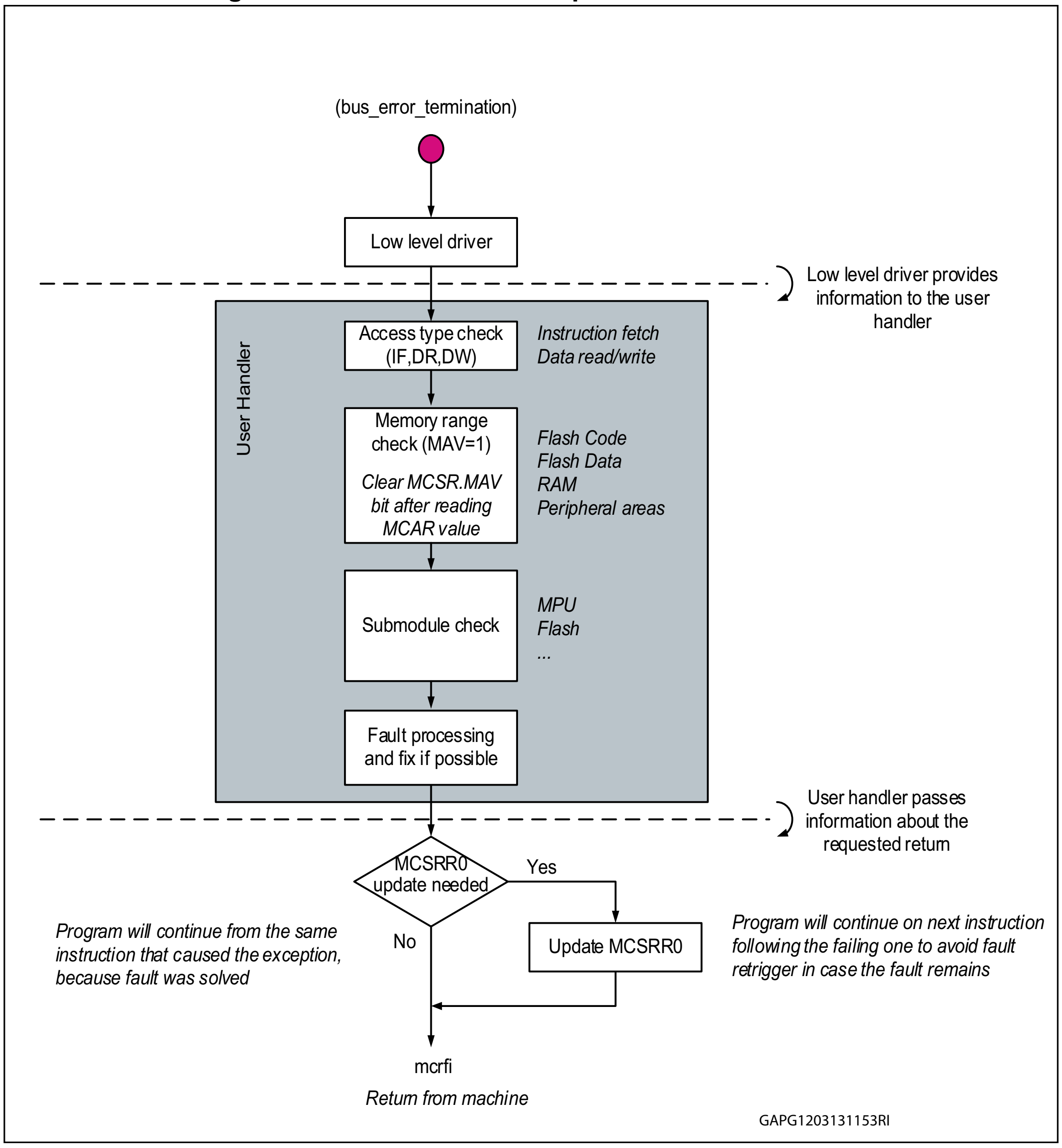
2.2 User handler

Here the root cause analysis is done. Such analysis requires supportive information from

- Low level driver (MCAR, MCSR etc.)
- Peripherals status registers for further elaboration

Based on the results of analysis and the corrective actions done, user handler should pass the information about the return type back to the low level driver; any modification to the MCSRR0 content should be made before *mcrfi* instruction.

Figure 3. Machine check exception user handler flow


(bus_error_termination)
Low level driver
Low level driver provides information to the user handler
User Handler
Access type check (IF,DR,DW)
Instruction fetch
Data read/write
Memory range check (MAV=1)
Clear MCSR.MAV bit after reading MCAR value
Flash Code
Flash Data
RAM
Peripheral areas
Submodule check
MPU
Flash
...
Fault processing and fix if possible
User handler passes information about the requested return
MCSRR0 update needed
Yes
No
Update MCSRR0
Program will continue from the same instruction that caused the exception, because fault was solved
Program will continue on next instruction following the failing one to avoid fault retrigger in case the fault remains
mcrfi
Return from machine
GAPG1203131153RI

3 SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx exception cases

This chapter lists the most common exception cases that application software can experience while running code on SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx devices.

Table 4. SPC564Axx/RPC564Axx/SPC56ELxx/RPC56ELxx exception causes

Exception cause	Error signaling	Exception	Description
Flash 2b ECC error	External bus error	Machine check	Two or multiple bit error in the Flash memory leads to the machine check exception when faulty area is read, instruction fetch or data read.

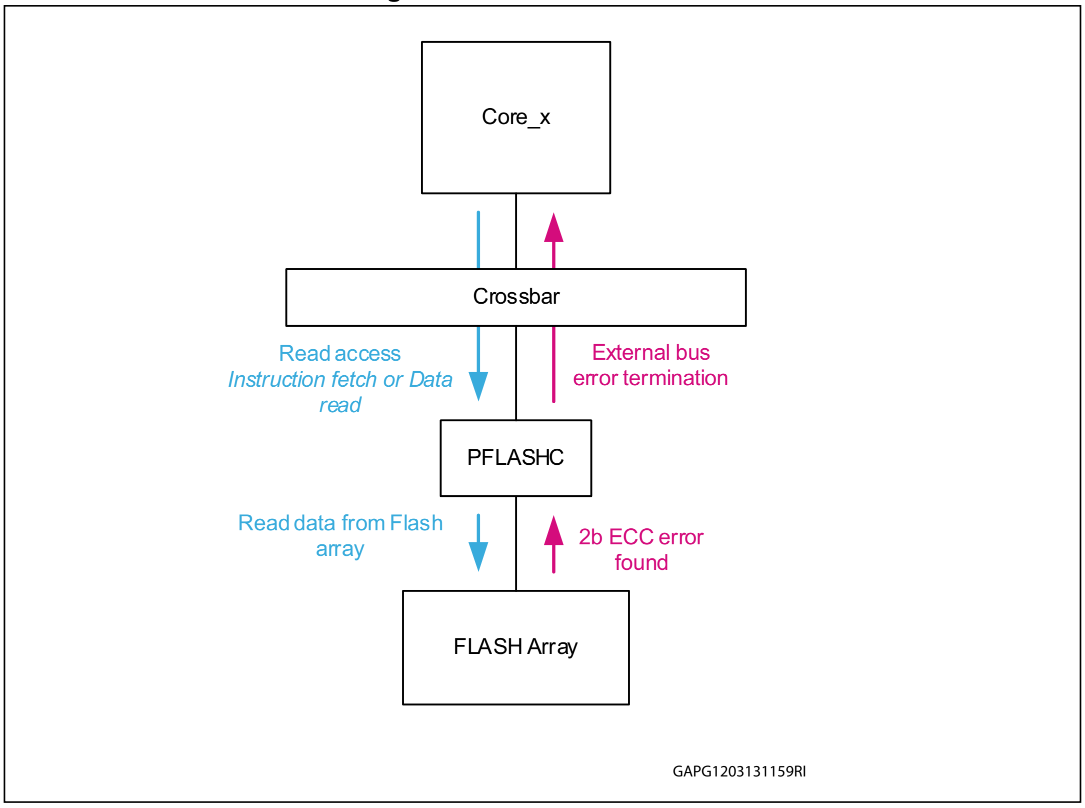
In general all protection access exceptions and 2b ECC exception lead to the same machine check exception because of external bus error termination. In such cases further analysis relies on memory area check.

3.1 Flash 2b ECC error

3.1.1 Cause of the exception

Platform flash memory controller (PFLASHC) terminates bus transaction between CPU and PFLASHC controller in case the Flash memory array signals 2b ECC problem during read access. This leads to machine check exception because of *bus_error* termination.

Figure 4. Flash 2b ECC error


Core_x
Crossbar
Read access
Instruction fetch or Data read
External bus error termination
PFLASHC
Read data from Flash array
2b ECC error found
FLASH Array
GAPG1203131159RI

3.1.2 Machine check exception status

Table 5. Flash 2b ECC - machine check exception status in core registers

Register	Description
MCSRR0	Address of the instruction that caused the exception. In case of ECC error in the data flash area, register modification is needed.
MCSR	Type of operation is highlighted here, instruction fetch, data load or data write.
MCAR	Target address that was accessed, but finished with 2b ECC error. This address can be used for further analysis.

3.1.3 Flash 2b ECC error detection by ECSM

Flash controller provides detection ability of ECC errors detection.

Table 6. Flash 2b ECC – ECSM registers related to ECC error detection

Register	Description
ECSM_ESR	The ECSM_ESR signals the last, properly enabled (in ECSM_ECR) memory event to be detected. RAM and Flash single bit errors, as well as dual bit errors, are signaled by separated status bits.
ECSM_ESR.R1BC	A reportable single-bit platform RAM correction has been detected.
ECSM_ESR.RNCE	A reportable non-correctable platform RAM error (2b ECC) has been detected.

Table 6. Flash 2b ECC – ECSM registers related to ECC error detection (continued)

Register	Description
ECSM_ESR.F1BC	A reportable single-bit platform flash correction has been detected.
ECSM_ESR.FNCE	A reportable non-correctable platform flash error (2b ECC) has been detected.

Maintaining of ECSM_ESR register to be performed properly.

For more details see *Section Appendix B: Reference documents*.

3.1.4 ECSM_ESR.FNCE implementation note for SPC564A70 device only

Flash controller always reads one complete prefetch buffer line (128-bit) from flash array. ECSM_ESR.FNCE bit detects ECC error **separately** for double word A (bit 0..63) and double word B (bit 64..127) in SPC564A70 device, and it can cause an unexpected behavior.

The following is an example to demonstrate it:
Assuming that an ECC error is present in upper word and lower word is accessed by core. Then the ECC error is detected during complete 128-bit line reading and core Machine check exception is invoked, but ECSM_ESR.FNCE bit is not set in this case. If Machine check exception handler tests the ECSM_ESR.FNCE bit only in our case, then it may unexpectedly assume that no ECC issue occurred.

Note: There is a possibility to check Flash_A.MCR.EER bit instead of the ECSM_ESR.FNCE. The implementation depends on the application needs.

For more details see *Section Appendix A: Comparison of microcontroller behavior during ECC error.*

3.1.5 ECSM_ESR implementation for SPC56ELx/RPC56ELx devices only

SPC56ELx/RPC56ELx devices have been designed with functional safety in mind. In case the reporting of dual bit errors is enabled in the ECSM, the device reacts in one of the safest way, i.e. a critical fault is triggered by the FCCU. The outcome of this critical fault is a functional reset of the device without any exception triggered to the core. Rationale for this severe reaction is that since the dual bit error cannot be corrected, software is not able to recover it. Then the safest reaction is assumed to be a reset.

Nevertheless this reset reaction prevents most of the flash EEPROM emulation drivers from working correctly. An ECC error is a standard error situation during read in flash area used for data EEPROM emulation. This situation is handled by the driver accordingly.

In case of a dual bit error, reporting is disabled in the ECMS and then a core exception is invoked instead of the reset. Core exception handler gives the possibility to the flash EEPROM emulation driver to react accordingly.

3.1.6 Flash 2b ECC error detection by Flash controller

Flash controller provides detection ability of ECC errors detection.

Table 7. Flash 2b ECC – flash controller registers related to ECC error detection

Register	Description
Flash_x.MCR.EER (Flash.MCR.ERR for SPC56ELxx/ RPC56ELxx devices)	EER provides information on previous reads. If a double bit detection occurred, the EER bit is set to a 1.This bit must then be cleared, or a reset must occur before this bit returns to a 0 state.
Flash_x.AR (Flash.ADR for SPC56ELxx/ RPC56ELxx devices)	The ADDR field provides the first failing address in the event of ECC event error (MCR[EER] set), single bit correction (MCR[SBC] set), as well as providing the address of a failure that may have occurred in a state machine operation (MCR[PEG] cleared). Note: Flash controller always reads one complete prefetch buffer line (128-bit) from flash array. The first failing address stored in the AR register could be anywhere inside the flash prefetch line address range and can differ from the address originally accessed.
x	x = A for SPC564A70 device x = A and/or B for SPC564A74, SPC564A80/RPC564A80 devices, because flash address range is covered by two flash modules flash_A and flash_B.

For more details see *Section Appendix B: Reference documents*.

3.1.7 Flash_x.MCR.ERR implementation note for SPC564A80/RPC564A80 device only

SPC564A80/RPC564A80 contains two different flash modules. Each of them contains its own Flash_x.MCR.EER bit. The decision to address a particular flash module depends on accessed address from a flash address range. For more details see *Section Appendix B: Reference documents*l.

Flash controller always reads one complete prefetch buffer line (128-bit) from flash array. Flash_x.MCR.EER bit is always set, if ECC error is detected in any of double word A (bit 0..63) or double word B (bit 64..127). In contrast, core Machine check exception is invoked only if the accessed one double word contains ECC error in SPC564A80/RPC564A80 device, and it can cause an unexpected behavior.

Th following is an example to demonstrate it:
Assuming that an ECC error is present in upper word and lower word is accessed by core. The ECC error is detected during complete 128 bit line reading and core Machine check exception is **NOT** invoked, but Flash_x.MCR.EER bit is set in this case (and it is not cleared automatically). Let us assume, that Machine check exception is invoked later on caused by another reason (e.g. due to memory protection by MPU). If the Machine check exception handler tests the Flash_x.MCR.EER bit only, then it may unexpectedly assume that flash ECC error has occurred instead. However the Machine check exception handler routine may handle this situation by cross-checking the data coming from FLASH AR and MCAR.

Note: There is also a possibility to check ECSM_ESR.FNCE bit instead of the Flash_x.MCR.EER registers. The implementation depends on the application needs.

For more details see *Section Appendix A: Comparison of microcontroller behavior during ECC error.*

3.1.8 User exception handler

Handler has to analyze the following:

- Type of access, instruction fetch, data read, and data write. Only instruction fetch or data read access is expected in case of 2b ECC Flash error.
- Memory range
  Memory access must be within the area belonging to the Flash memory. User has to know which part belongs to the code flash and which part belongs to the data flash memory.

3.1.9 Error solving

Flash 2b ECC error can be solved only with erase of the flash sector containing the cell with 2b ECC error. It is usually not done in the exception handler itself, because it takes a significant amount of time.

The decision on what to do in case of 2b ECC error is application specific. Whether to go to degraded mode or to continue (i.e. the case of EEPROM emulation) and solve the issue later in the application.

If the decision is to continue, user handler has to request a modification of the MCSRR0 register to continue the program flow with th next instruction. Otherwise the program would be stuck in the reading of the fault flash address invoking machine checks.

Appendix A Comparison of microcontroller behavior during ECC error

Table 8 is a summary of the behavior of different 90nm microcontrollers in case of either single or double bit error in the flash. To understand the table, the user should keep in mind that flash is accessed in word line of 128-bit. Each time a master would like to access (read / code fetch) a location, which belongs to a certain word line, the whole word line is read out of the flash. The word line consists of two double-words A and B. Each double-word contains its own ECC. Even there is always read complete word line, behavior of microcontroller can differ regarding which double-word is addressed and which one contains an ECC error.

One concrete word line starting at address 0x00030000 is chosen as an example.

The first 2 columns of the table represent the address location which can be accessed and can be affected by single/double bit error. Each row represents a combination of access and an ECC error:

- The marked cells in the first 2 columns are affected by an ECC error.
- Cells with the text ‘Accessed by master’ are actually accessed by one of the crossbar master.
- In addition, few cells are marked to highlight some differences in terms of behavior.

The other columns contain reactions of selected registers, - separately for each microcontroller.

Table 8. Summary of reactions to single/double bit error

FLASH line (128bit) e.g. ADDRESS = 0x30000 means addr.range: 0x30000..0x3000F		SPC564A70				SPC564Axx/RPC564Axx				SPC563M				SPC56EL/RPC56EL family				SPC560P family			
double word A e.g. addr. range: 0x30000.. 0x30007	**double word B e.g. addr. range: 0x30008.. 0x3000F**	**FLASH MCR EER**	**FLASH AR**	**ECSM ESR FNCE**	**IVOR1 EXCEPTION /**	**FLASH MCR EER**	**FLASH AR**	**ECSM ESR FNCE**	**IVOR1 EXCEPTION /**	**FLASH MCR EER**	**FLASH AR**	**ECSM ESR FNCE**	**IVOR1 EXCEPTION /**	**FLASH MCR EER**	**FLASH AR**	**ECSM ESR FNCE**	**BUS ERROR /**	**FLASH MCR EER**	**FLASH AR**	**ECSM ESR FNCE**	**BUS ERROR / RESET**
	Accessed by master	0	No change	0	NO	0	No change	0	NO	0	No change	0	NO	0	No change	0	NO	0	No change	0	NO
Accessed by master		0	No change	0	NO	0	No change	0	NO	0	No change	0	NO	0	No change	0	NO	0	No change	0	NO
	Accessed by master	1	0x30000	0	YES	1	0x30000	0	NO	1	0x30000	1	YES	1	0x30000	0	NO	1	0x30000	1	YES
Accessed by master		1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES
	Accessed by master	1	0x30008	1	YES	1	0x30008	1	YES	1	0x30008	1	YES	1	0x30008	1	YES	1	0x30008	1	YES
Accessed by master		1	0x30008	0	YES	1	0x30008	0	NO	1	0x30008	1	YES	1	0x30008	0	NO	1	0x30008	1	YES

Table 8. Summary of reactions to single/double bit error (continued)

FLASH line (128bit) e.g. ADDRESS = 0x30000 means addr.range: 0x30000..0x3000F		SPC564A70				SPC564Axx/RPC564Axx				SPC563M				SPC56EL/RPC56EL family				SPC560P family			
double word A e.g. addr. range: 0x30000.. 0x30007	double word B e.g. addr. range: 0x30008.. 0x3000F	FLASH MCR EER	FLASH AR	ECSM ESR FNCE	IVOR1 EXCEPTION /	FLASH MCR EER	FLASH AR	ECSM ESR FNCE	IVOR1 EXCEPTION /	FLASH MCR EER	FLASH AR	ECSM ESR FNCE	IVOR1 EXCEPTION /	FLASH MCR EER	FLASH AR	ECSM ESR FNCE	BUS ERROR /	FLASH MCR EER	FLASH AR	ECSM ESR FNCE	BUS ERROR / RESET
	Accessed by master	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES
Accessed by master		1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES	1	0x30000	1	YES

Note: *Marking in the first 2 columns of the table represents the address locations which are affected by an ECC error.*

Accessed by the master means address location accessed (read or write or core instruction fetch) by crossbar (XBAR) master. Microcontroller core is only one of the XBAR masters.

Appendix B Reference documents

- Z4d Core Reference Manual
- *SPC56EL60 32-bit MCU family built on the embedded Power Architecture®* (RM0032, Doc ID 15265)
- *SPC56XL70xx 32-bit MCU family built on the embedded Power Architecture®* (RM0042, Doc ID 023986)
- *SPC564A74xx, SPC564A80xx 32-bit MCU family built on the embedded Power Architecture®* (RM0029, Doc ID 15177)
- *SPC564A70B4, SPC564A70L7 32-bit MCU family built on the embedded Power Architecture®* (RM0068, Doc ID 18132)

B.1 Acronyms

Table 9. Acronyms

Acronym	Name
ECC	Error Correction Code
EDC	Error Detection Code
2b ECC	double bit error (it is only detected by the ECC/EDC hardware)
1b ECC	Single bit error (it’s detected and correct by the ECC/EDC hardware.
NMI	Non maskable interrupt

Revision history

Table 10. Revision history

Date	Revision	Changes
12-Dec-2013	1	Initial release
19-Dec-2013	2	Modified *Table 8*.
06-Oct-2015	3	Robust root part numbers added.