AN4367
Application note

SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx device exception handling

Introduction

This document provides an overview of SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx exception handling with a main focus on different kind of exceptions that the application code can face during the runtime like Flash 2b ECC error, RAM 2b ECC error, MPU protection violation, AIPS access protection violation and others.

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

The next section, lists the SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx exceptions, describes the reason of the exception, how to find it and the options available to remove the fault.

Contents

List of tables

List of figures

1 Z4 Core exception overview

Z4 Core used on SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx 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 detail list of exception causes and their mapping to interrupt handlers can be found in the Z4 Core Reference Manual. This reference manual is available online at http://www.st.com.

This chapter gives a simple overview of the machine check interrupt that is utilized for several important fault states of SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx device.

1.1 Machine check interrupt (IVOR1)

Machine check interrupt is a handler that services multiple fault events that can 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 SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx device, like MPU protection fault, 2bECC error in the Flash or RAM memory etc. The reason is that most of the faults are signaled back as 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. These registers are used to find the source of the exception and based on it to decide how to solve it.

Table 2. Machine check register

Register	Content
MCSR (syndrome register)	Register indicates the source of machine check condition that gives possibility to differentiate between them
MCAR (address capture)	Register contains for 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.

Table 2. Machine check register (continued)

Register	Content
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 as it collects 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 the exception is always taken whenever the condition occurs. They enable to differentiate between Instruction fetch, Data store and load.
Non-maskable interrupt (NMI)	Not MSR_{ME} gated exception that 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 have to be enabled by MSR_{ME} bit. They are cumulative. This machine check exception group triggers capture of the corresponding address to the MCAR register 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 the 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 in the beginning of the machine check interrupt. It stores the address of the instruction that caused the error condition.

It is used in 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

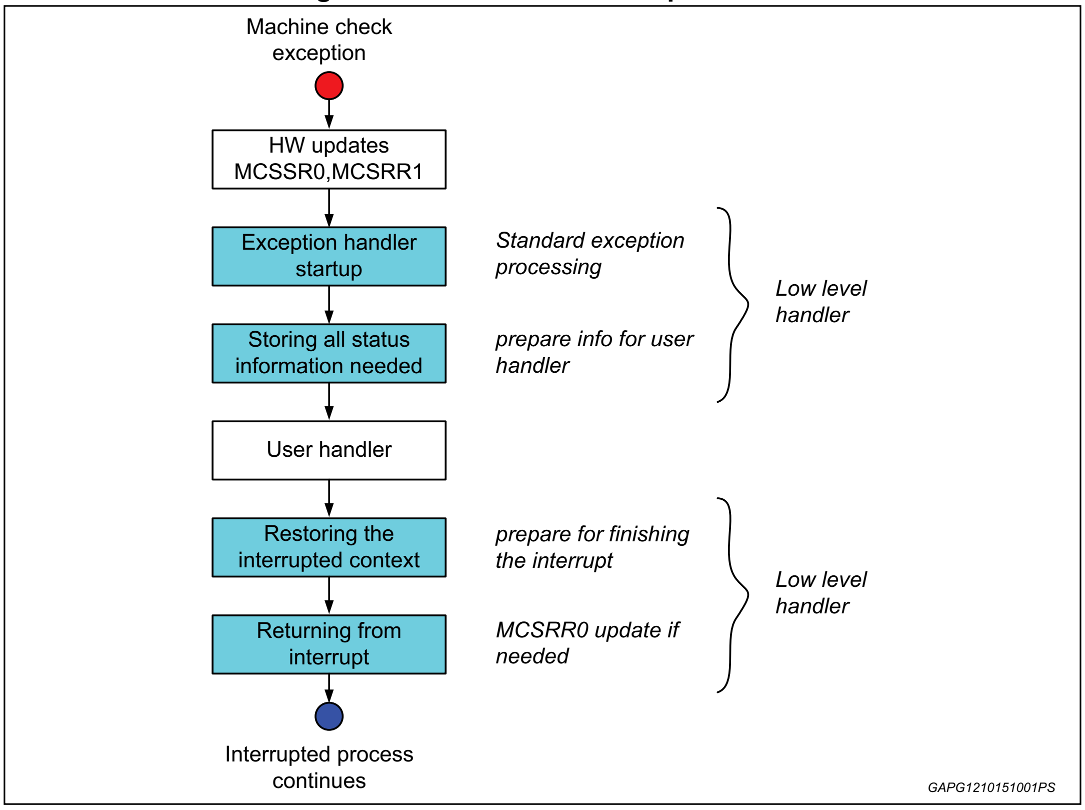
2.1 Low level handler

Low level handler is responsible for the first and last part of the exception execution. It is usually written in assembly as it needs to execute the proper instruction sequence before it can pass the code execution to a 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 interrupted 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
GAPG1210151001PS

2.1.1 Start phase

The first steps when a machine check exception occurs are done by the hardware (Core) which stores the content of the MSR register and the address of the current instruction pointer whenever it is possible (precise exception).

Low level driver immediately starts processing after. It executes several steps like machine check status register saving, context of the interrupted process saving and others. This part should store some additional information too as it will be used by higher layer user handler to analyze the root cause of the exception later.

A detailed description of the machine check resources, their meaning and proper handling in case of interrupt are described in the Z4 Core User Manual documentation. The low level handler shall follow rules and recommendations described there.

2.1.2 Final phase

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

Before *rfmci* 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, such information should be decided in the user handler and passed down to the low level driver

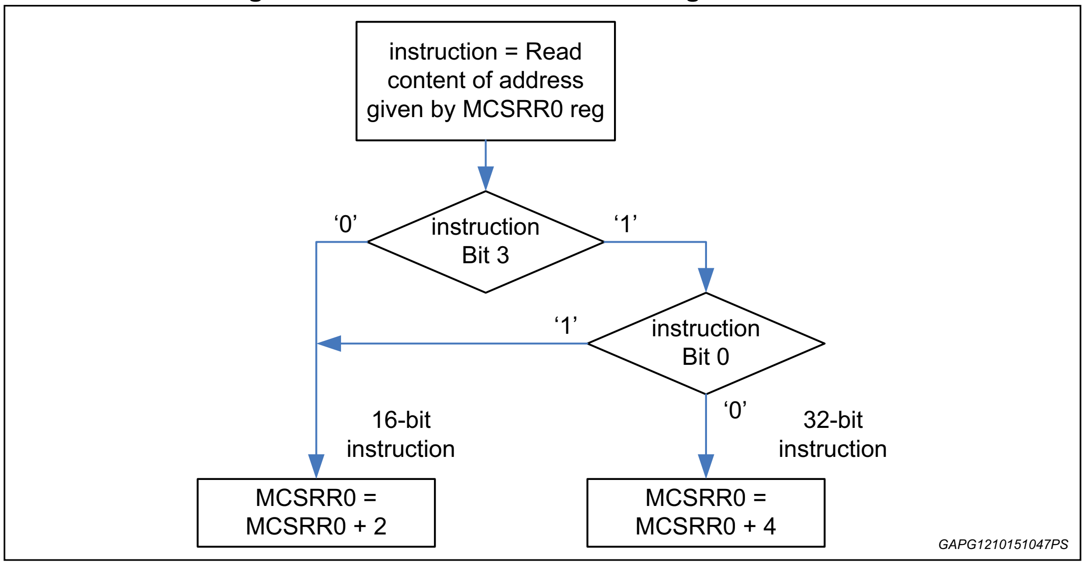
1. User handler was able to find the cause of the machine check exception and to fix it in a way, that program can re-execute the same instruction that caused the machine check exception
2. User was able to find the cause of the exception, but the problem remains and re-executing the same instruction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

2.1.3 Modification of the MCSRR0 register

In case the cause of the exception cannot be removed, MCSRR0 register value has to be 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 increment the value 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



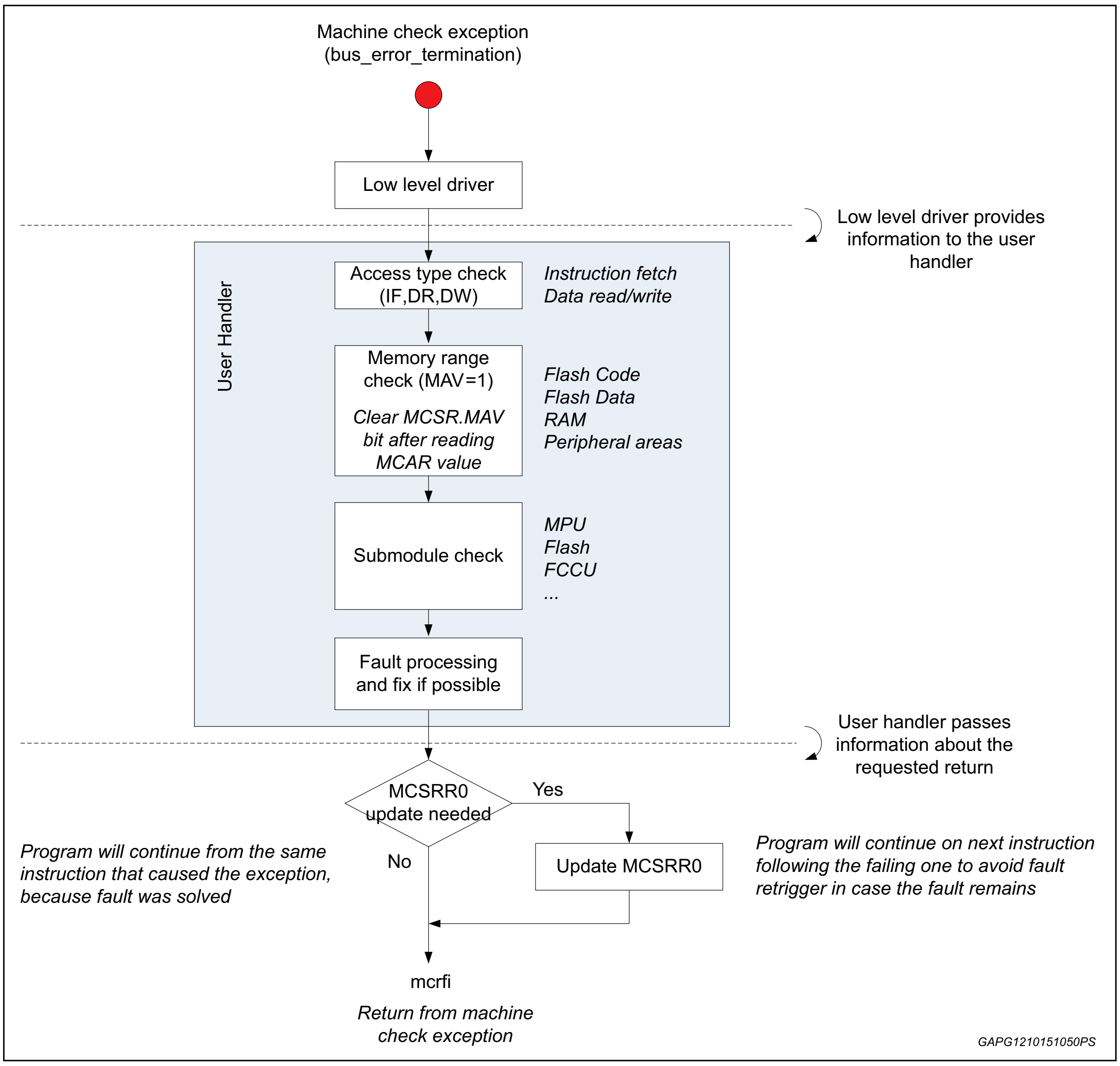
2.2 User handler

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

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

Based on the results of analysis and corrective actions done, user handler should pass the information about the return type back to the low level driver; indication if MCSRR0 content should be modified or not before *mcrfi* instruction.

Figure 3. Machine check exception user handler flow


Machine check exception
(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
FCCU
...
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 check exception
GAPG1210151050PS

3 SPC56EL60xx/RPC56EL60xx/ SPC56xL70xx/RPC56xL70xx exception cases

This chapter lists the most common exception cases that application software can experience while running code on SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx device exception handling device.

Table 4. SPC56EL60xx/RPC56EL60xx/SPC56xL70xx/RPC56xL70xx 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.

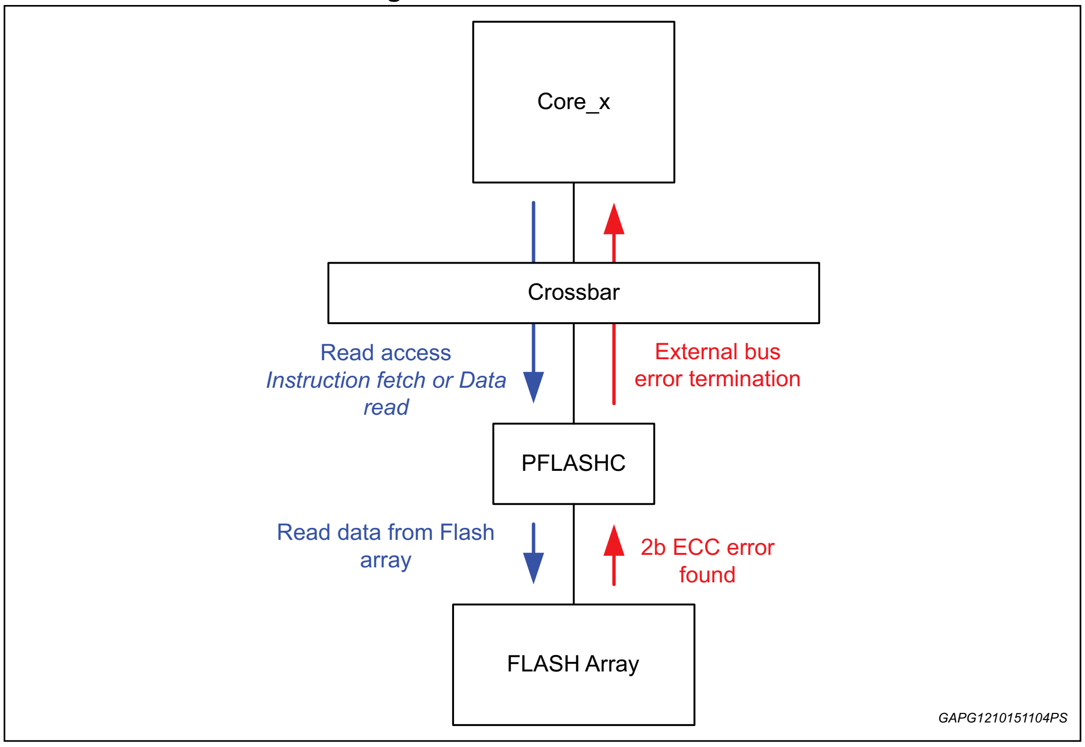
In general all protection access exceptions and 2b ECC exception leads to the same machine check exception because of external bus error termination. In such case further analysis relies on memory area check.

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
GAPG1210151104PS

3.1.2 Machine check exception status

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

Register	Description
MCSRR0	Address of the instruction that caused the exception. In case of ECC error in the data flash area, register modification will be most probably 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 User exception handler

Handler has to analyze:

- Type of access, instruction fetch, data read, data write. Only instruction fetch or data read access is expected in case of 2bECC 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 to the data flash memory.

3.1.4 Error solving

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

It is application specific the decision on what to do in case of 2b ECC error, if to go to a degraded mode or to continue, the case of EEPROM emulation, and to solve the issue later in the application.

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

Revision history

Table 6. Document revision history

Date	Revision	Changes
02-Oct-2013	1	Initial release.
15-Oct-2015	2	Robust root part numbers added.