AN4276
Application note

ECC management on SPC560x

Introduction

This document describes Error Correction Code (ECC) management and implementation on SPC560x microcontroller family. It describes both hardware and software aspects linked to the ECC mechanism used to protect content of memories.

It starts with an overview of ECC protection and than it jumps to more detailed hardware ECC Fault management and how it is implemented in the microcontroller. The hardware notification on one side is important but not enough to correctly manage memory content faults in the application. That is why this note takes closer look on software aspects in its second part.

Contents

List of tables

List of figures

1 ECC overview

Error correction codes, ECC, are known techniques to protect data information against single or multiple bit errors. SRAM memories represent main field of ECC usage, where soft induced bit flips errors are common and where these problems increase with processes going to submicron ranges. ECC protection is one of the highly recommended measures of memory protection in high reliable and safety systems.

There are several conventional algorithms used for ECC protection, single-error-correcting (SEC) Hamming code, single-error-correcting-double-error-detecting (SEC-DED) modified Hamming, or SEC-DED Hsiao code. All of these algorithms are based on principle of extending information bits with so called check bits in a way that bit error on reception is identified and corrected in case of one bit error or detected in case of two errors.

1.1 ECC principle

Set of information symbols (data), where any bit error in one symbol leads to another valid, but wrong symbol, is transformed to another set of code words by extending information bits with check bits. The size of new code set is higher than the size of original one. It gives an option to map original information symbols to code words with specific characteristic like Hamming distance specifying number of bits in which these code words differ.

1.2 Number of check bits

There is a minimum number of check bits that are needed for SEC-DED protection. *Table 1* shows relations between data width and number of check bits.

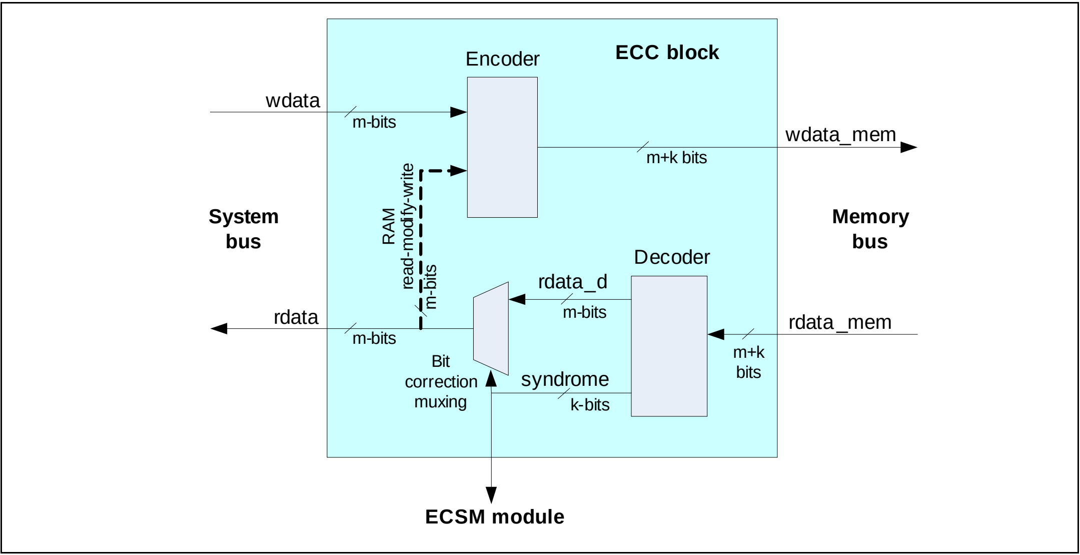
Table 1. SEC-DED code - number of check bits needed

Data width	Number of check bits
16	6
32	7
64	8

2 ECC on SPC560x microcontroller family

SPC560x microcontroller family implements ECC mechanism for all its memories, RAM, Code Flash and Data Flash. Each memory has its own ECC block that calculates parity bits with each write access and checks data validity with each read access where 1-bit correction is done automatically if needed. Decode result is reported out of the ECC module and collected in ECSM module and in parallel in Flash modules for Flash ECC errors. General scheme is shown in *Figure 1*.

Figure 1. ECC block scheme



2.1 RAM ECC implementation

Data are protected on 32-bit word boundaries with seven check bits. It means that 39-bits are stored on write access and checked on read access.

7-bit check bits are calculated for whole 32-bits that have consequence for 8 or 16-bit accesses where the content of the requested address is read and checked first then merged with modified bytes, then ECC is calculated for whole 32-bit word and written back to memory.

From that reason of read-modify-write operations, access time can vary depending on current and previous operation as shown in *Table 2*.

Table 2. SRAM timing

Current operation	Previous operation	Number of wait states required
Read	Idle / Pipelined read	1
	8/16/32 bit write	0 (same address)
		1 (different address)

Table 2. SRAM timing (continued)

Current operation	Previous operation	Number of wait states required
Pipelined read	Read	0
8/16/ bit write	Idle / Read	1
	Pipelined 8/16 bit write 32 bit write	2
	8/16 bit write	0 (same address)
Pipelined 8/16/32 bit write	8/16/32 bit write	0
32 bit write	Idle / 32 bit write / Read	0

ECC block has impact on SRAM initialization after power-up or destructive resets where ECC check bits may not correspond to data content and has to be recalculated. It means that whole memory must be written with 32-bit write accesses. Otherwise read access to ECC non-initialized area will trigger ECC exceptions.

2.2 Code and Data Flash memory ECC implementation

Differently from RAM, Flash memories are protected on 64-bit word boundaries with eight check bits. Because of the nature of the Flash memory that enables programming memory bits only in one direction from logical one to logical zero whole 64-bits should be programmed at once. Reprogramming same word with different value may lead to different ECC check bit pattern that would cause reprogramming error.

Such limitation of bit reprogramming in the same location widely used in EEPROM emulation can be a problem. Therefore ECC scheme implemented in Flash memories is slightly modified to support such features but in a limited manner. There are few data patterns selected from the whole set of available codes that do not lead to change of ECC check bit vector, see *Table 3*. Using such words can be used in EEPROM emulation software to change status of data sets or sector status information in the same address location.

Table 3. Flash codes with the same syndrome

Double word
0xFFFF_FFFF_FFFF_FFFF
0xFFFF_FFFF_FFFF_0000
0xFFFF_FFFF_0000_0000
0xFFFF_0000_0000_0000
0x0000_0000_0000_0000

Each Flash module contains status bits reflecting ECC check result for each read access. ECC hardware sets these bits whenever there is an ECC error. They are cleared from application software. Seeing status bit set means that there was an error in one of the read memory accesses occurring between last and current status bit check.

2.3 ECSM module

A part of Error Correction Status Module is dedicated to collecting ECC information reported by memories and underlying bus system. It provides set of registers storing extended information about last ECC fault signaled by status register ESR.

ECSM module differentiates between RAM and Flash memories, where Code and Data Flash memories are merged and signaled as one Flash ECC fault. Code analyzing from which memory the fault comes has to use FEAR register storing the address and to bind it to the correct Flash region.

Table 4. ECSM module register set

Register	Description	Register	Description
ECR	ECC Configuration	FEDR	Flash ECC Data
ESR	ECC Status	REAR	RAM ECC Address
EEGR	ECC Error Generation	RESR	RAM ECC Syndrome
FEAR	Flash ECC Address	REMR	RAM ECC Master
FEMR	Flash ECC Master Number	REAT	RAM ECC Attributes
FEAT	Flash ECC Attributes	REDR	RAM ECC Data

3 ECC error hardware exceptions

There are two types of ECC fault notifications that are generated by hardware of SPC650x microcontroller.

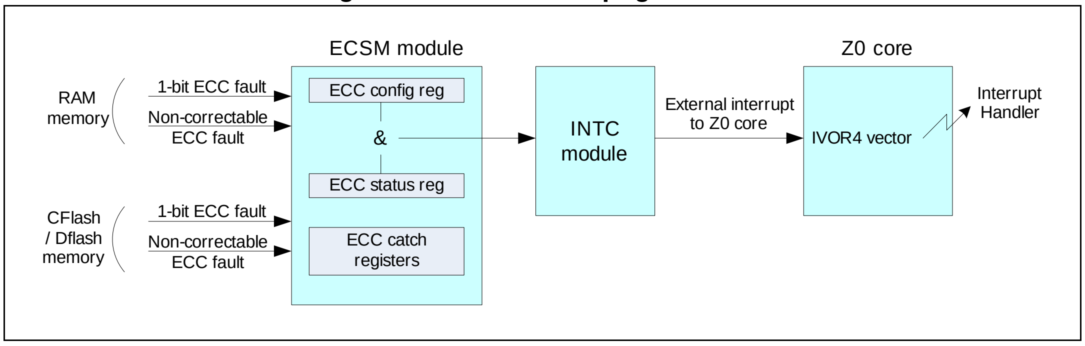
- ECSM module interrupts
- Exceptions events of the core

3.1 ECSM module interrupts

ECSM module interrupt can be generated in the case of both kinds of ECC faults, 1-bit and 2-bit, and they are further distinguished for RAM and Flash memory. This interrupt belongs among external peripheral interrupts that are linked to IVOR4 vector. *Figure 2* shows block scheme of ECSM interrupt routing in the microcontroller.

All interrupts that can be generated by ECSM module are configurable and can be masked by the user in ECR register.

Figure 2. ECSM interrupt generation



3.2 Core exceptions

In all cases, non-correctable ECC errors that occur during instruction fetch or data read cause generation of core exceptions because of bus transaction errors between Z0 core and given memory target. Core exception events cannot be masked and should be handled properly to solve the issue.

Several exceptions can be generated based on type of memory access, instruction fetch or data fetch, and configuration of machine state register MSR, see *Table 5*.

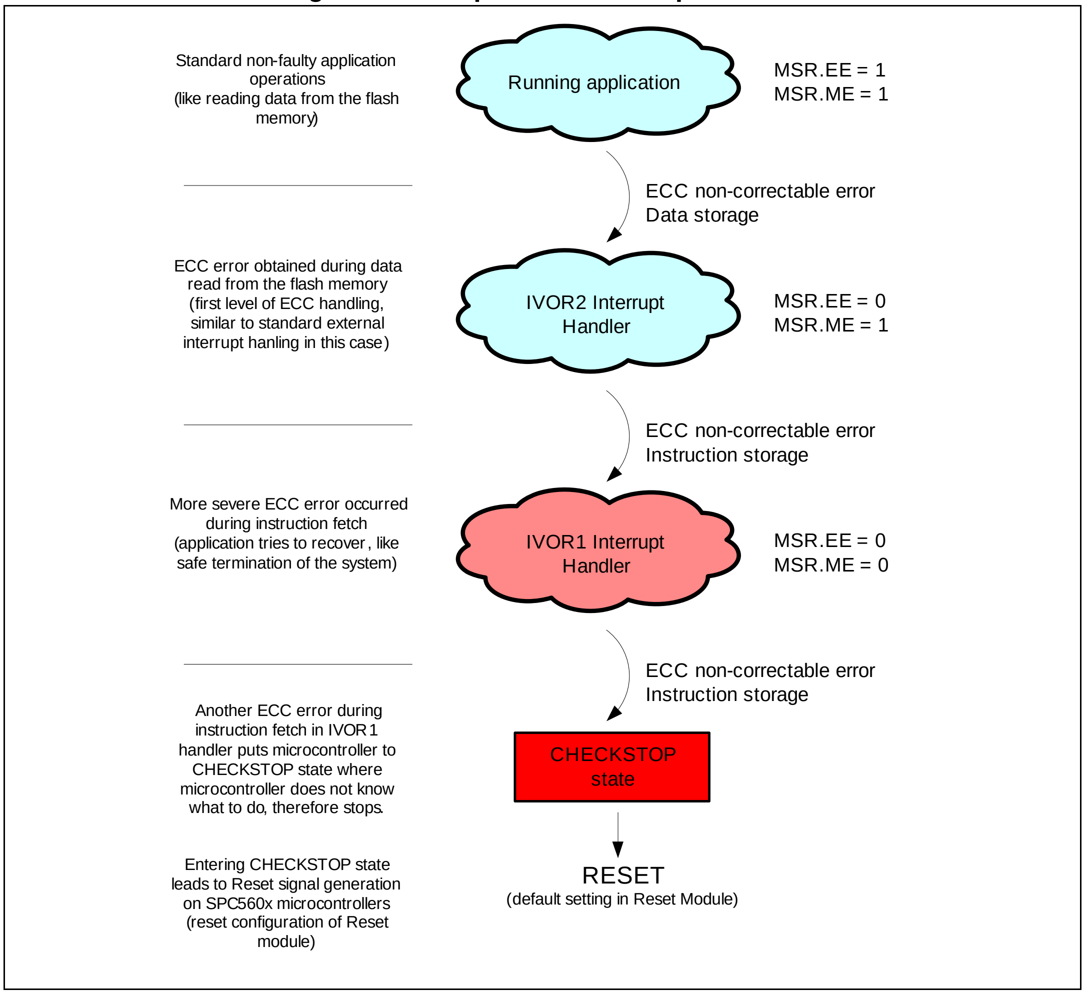
Single bit interrupts does not generate core exceptions, they can be caught by enabled ECSM interrupts. Not all non-correctable ECC errors are signaled by core exceptions like ECC error during discarded not taken branch fetch, but they are signaled by ECSM module.

Table 5. Non-correctable ECC exceptions

MSR register bits		Access type	Result / exception
EE	ME		
0	0	Instruction or data	Enter Checkstop state
0	1	Instruction or data	IVOR1 (Machine check interrupt)
1	x	Data	IVOR2 (Data storage interrupt)
1	x	Instruction	IVOR3 (Instruction storage interrupt)

From the *Table 5* it is visible that ECC core exception system provides three priority notification levels. Application has several trials to overcome the ECC error if it is possible. *Figure 3* shows an example of such exception flow.

Figure 3. Example of core exception flow


Standard non-faulty application operations (like reading data from the flash memory)
Running application
MSR.EE = 1
MSR.ME = 1
ECC non-correctable error Data storage
ECC error obtained during data read from the flash memory (first level of ECC handling, similar to standard external interrupt hanling in this case)
IVOR2 Interrupt Handler
MSR.EE = 0
MSR.ME = 1
ECC non-correctable error Instruction storage
More severe ECC error occurred during instruction fetch (application tries to recover, like safe termination of the system)
IVOR1 Interrupt Handler
MSR.EE = 0
MSR.ME = 0
ECC non-correctable error Instruction storage
Another ECC error during instruction fetch in IVOR1 handler puts microcontroller to CHECKSTOP state where microcontroller does not know what to do, therefore stops.
CHECKSTOP state
Entering CHECKSTOP state leads to Reset signal generation on SPC560x microcontrollers (reset configuration of Reset module)
RESET
(default setting in Reset Module)

4 ECC exception handling

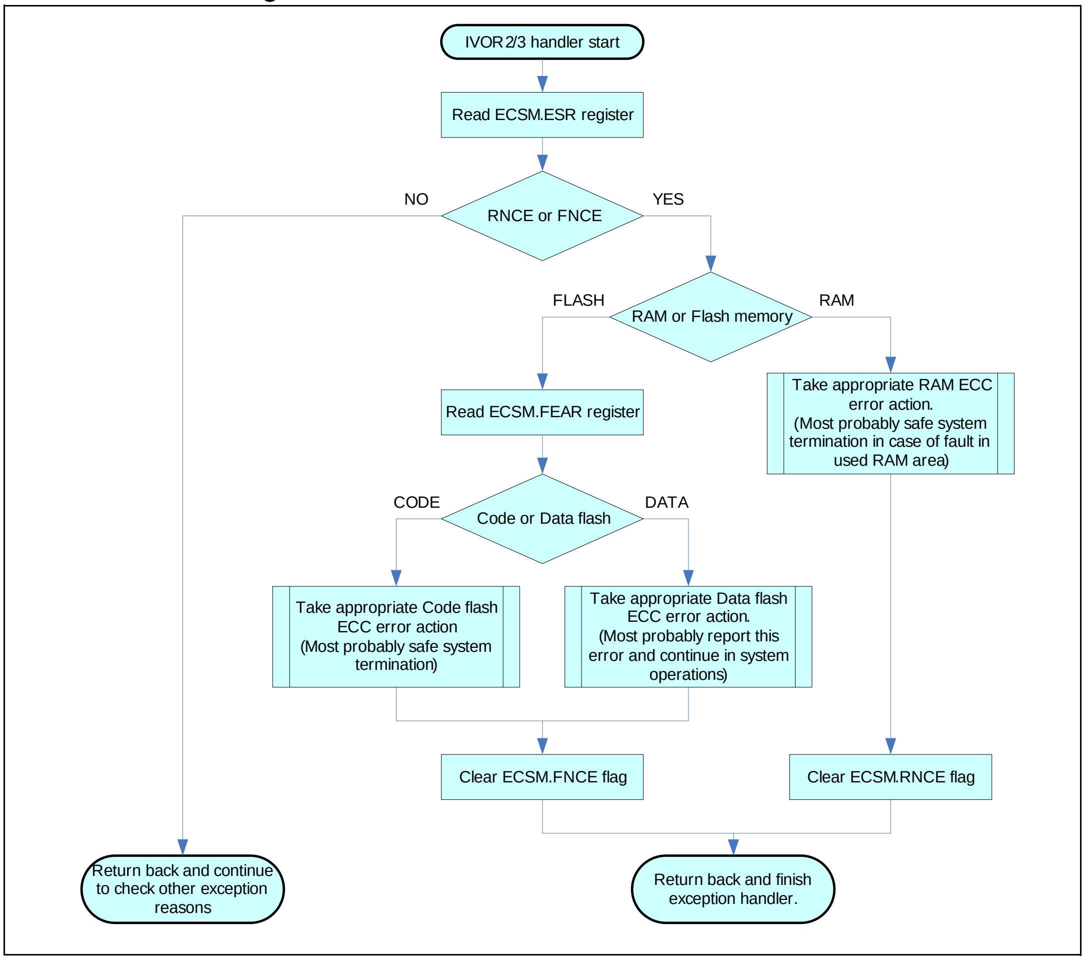
As seen above, microcontroller always generates one of the exceptions for non-correctable ECC errors depending on the EE and ME bits setting in the MSR register. Proper software handlers should be placed on IVOR2, IVOR3 and IVOR1 exception vectors.

Z0 core used on SPC560x microcontrollers adds additional complexity to the exception handling mixing several different exception causes to one vector, which means that software handler has to find the reason of the interrupt first. IVOR2 and IVOR3 exceptions merge following causes

- ECC non-correctable errors (IVOR2 and IVOR3)
- MPU access faults (IVOR2 and IVOR3)
- Register protection violation (IVOR2)

General software flow of the exception handler IVOR2 and IVOR3 with focus on ECC processing is shown in *Figure 4*. IVOR1 can differ in a way that having this exception (machine check) means that system recovery is hardly possible and system should safely terminate.

Figure 4. General IVOR2/3 handler software flow


IVOR 2/3 handler start
Read ECSM.ESR register
NO
RNCE or FNCE
YES
FLASH
RAM or Flash memory
RAM
Read ECSM.FEAR register
Take appropriate RAM ECC error action. (Most probably safe system termination in case of fault in used RAM area)
CODE
Code or Data flash
DATA
Take appropriate Code flash ECC error action (Most probably safe system termination)
Take appropriate Data flash ECC error action. (Most probably report this error and continue in system operations)
Clear ECSM.FNCE flag
Clear ECSM.RNCE flag
Return back and continue to check other exception reasons
Return back and finish exception handler.

4.1 RAM ECC error action

It is up to the system requirements what to do in case of non-correctable ECC error during read access to RAM memory. Handler can decide upon analysis of additional ECSM registers, like ECSM.REAR register determining RAM address. If the application considers such event as critical it is up to the application to terminate the system in the managed way. To clear non-correctable ECC error in the memory means to overwrite whole 32-bit word of affected RAM cell with a new value that refreshes data and ECC value.

4.2 Code Flash ECC error action

Criticality of non-correctable ECC error from Code Flash memory depends on type of the access. Error during the instruction fetch represents critical situation and can be taken as critical event requiring system termination.

ECC error during data access can or cannot represent critical event depending on system implementation, like data redundancy etc. In such case error action depends on system design.

4.3 Data Flash ECC error action

Non-correctable ECC exception coming from data Flash accesses can happen regularly and may not represent critical events especially when data Flash memory is used for EEPROM emulation. In such case it is desired to continue the program operation and let application software to solve the ECC data error.

In principle there are two scenarios how error handler, IVOR2 in this case, can let application software to continue its flow.

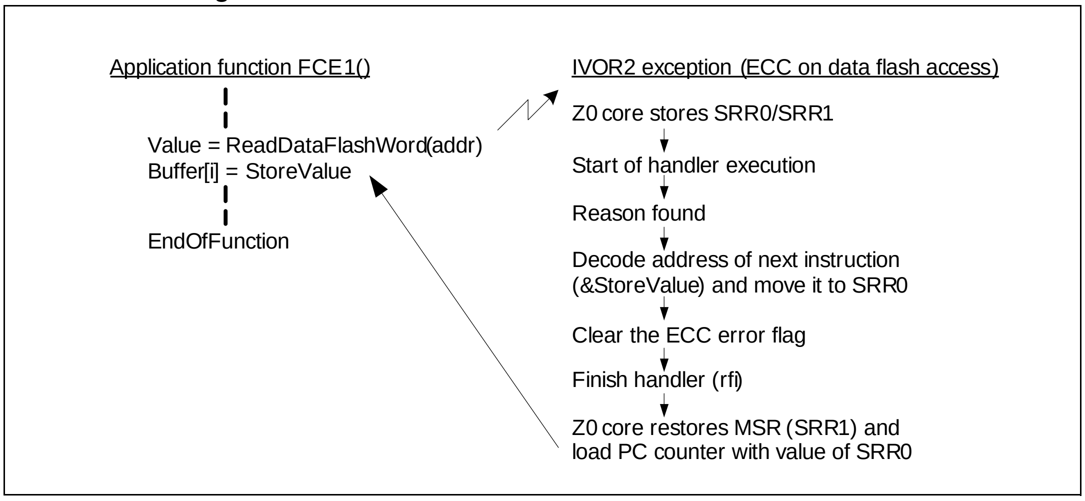
- Continue with successive instruction of the read function
- Continue from known address in the read function

Both scenarios are based on SRR0 register modification which changes effective address where to continue once the handler returns back by *rfi* instruction. Modification of SRR0 register is necessary because it is loaded by effective address of the instruction that caused the exception, which means that using the original value of SRR0 would cause execution of the same instruction accessing corrupted memory cell once returning from the handler.

4.3.1 Continue with successive instruction of the read function

Target is to continue with the instruction that follows the instruction reading the data from Flash memory like in *Figure 5*.

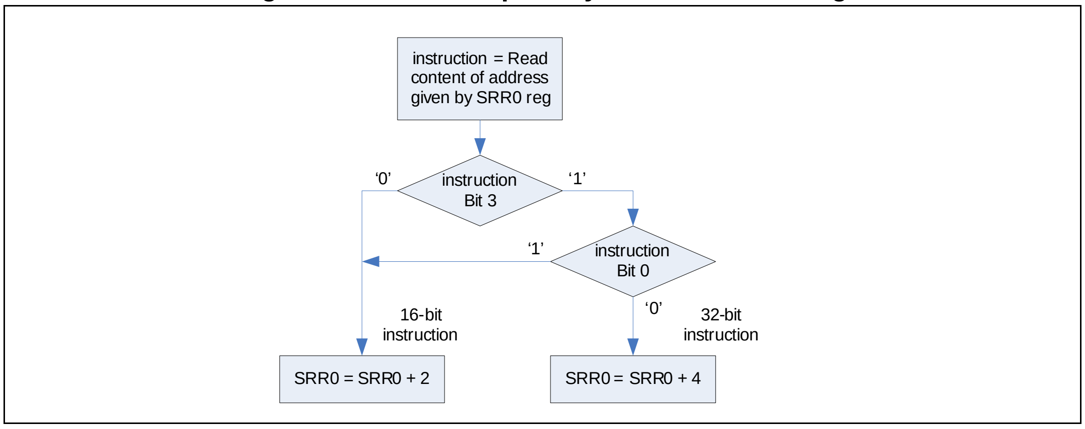
Figure 5. Pseudo-Code for next instruction handler flow


Application function FCE1()
Value = ReadDataFlashWord(addr)
Buffer[i] = StoreValue
EndOfFunction
IVOR2 exception (ECC on data flash access)
Z0 core stores SRR0/SRR1
Start of handler execution
Reason found
Decode address of next instruction (&StoreValue) and move it to SRR0
Clear the ECC error flag
Finish handler (rfi)
Z0 core restores MSR (SRR1) and load PC counter with value of SRR0

Because of variable length instruction encoding, handler has to decode the length of the instruction on the address stored in SRR0 register. It is done by means of analyzing OPCODE of the instruction that can reveal the actual length, 16 or 32-bit. Once the length is known value of SRR0 will be increased by the computed length.

Algorithm to find the instruction length for Z0 core is shown in *Figure 6*, where first six bits of the first byte represents a primary OPCODE. Bits are in big endian form, where the bit 0 in the leftmost bit in the word.

Figure 6. Instruction primary OPCODE decoding



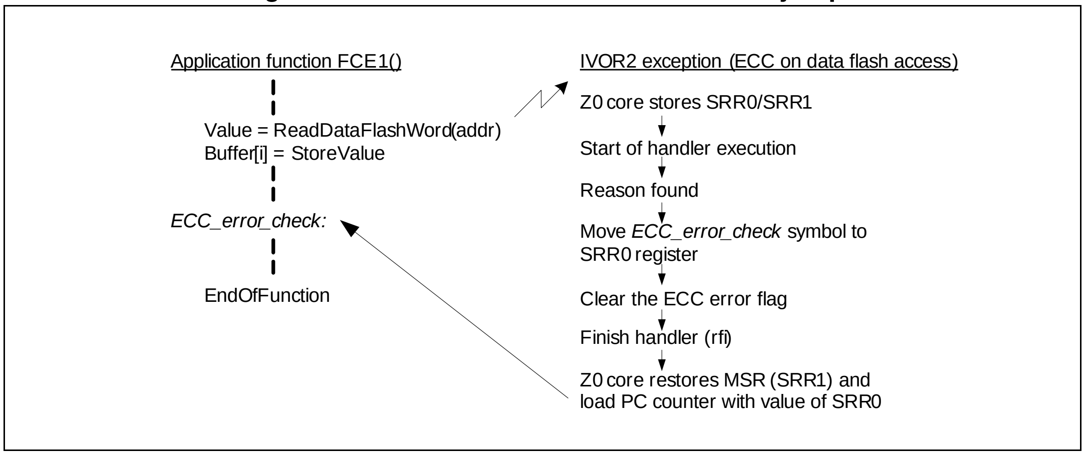
Next instruction decoding approach has an advantage that it returns exactly to next instruction in the application function which caused the ECC error regardless which function caused the error. Later in the function code can be check of data validity.

Disadvantage can be processing time and exception load if for example function reads multiple data bytes in the loop and checks for the error after that. Such loop will generate ECC exception for each read where the data are corrupted.

4.3.2 Continue from known address in the read function

It is similar to the previous approach of next instruction decode, but return from the error handler is to a known symbol in the function that can cause the ECC data access fault without need of instruction decode like in *Figure 7*.

Figure 7. Pseudo Code for known address jump


Application function FCE1()
Value = ReadDataFlashWord(addr)
Buffer[i] = StoreValue
ECC_error_check:
EndOfFunction
IVOR2 exception (ECC on data flash access)
Z0 core stores SRR0/SRR1
Start of handler execution
Reason found
Move ECC_error_check symbol to SRR0 register
Clear the ECC error flag
Finish handler (rfi)
Z0 core restores MSR (SRR1) and load PC counter with value of SRR0

An advantage of this approach is that ECC error is immediately found and possible read accesses stopped. This can save some time and exception load when reading long buffers without immediate error check.

The disadvantage is that there must be only one function that can cause ECC error. It means that in whole application there is just one place where data Flash is read. It is strongly dependent on software implementation.

4.4 Clearing error flags

ECC error flags in the ECSM module should be cleared for recoverable scenarios in order to enable proper exception handling for other causes, like register protection etc. Other reason for clearing ECSM status bits is the property of the ECSM behavior where new ECC error overwrites the old information, because there is only one set of registers storing extended information. The recommendation is to clear these flags always, even in the case of non-recoverable situations.

5 Application software notes

This chapter discusses some application aspects and areas that are linked to ECC protection mechanism implemented on SPC560x microcontrollers.

5.1 RAM memory checks

SEC-DED ECC mechanism provides meaningful protection against soft errors. But to use benefits of the ECC protection application support is needed. The list bellow shows the main points.

- Configure ECSM module to report 1-bit RAM ECC errors
- Enable interrupt handling from ECSM module
- Read regularly content of the RAM memory, that triggers ECC check
- Correct any single bit error caught by ECC in the interrupt

Each 1-bit ECC error in the RAM memory shall be caught and corrected by the application. The reason is to prevent accumulation of several 1-bit errors that can lead to non-correctable error with potential undesired system shutdown.

Corrective action is based on writing back the corrected data value to the affected address. Needed information is stored in ECSM registers, where affected address cell is given by REAR register and corrected value by REDR register.

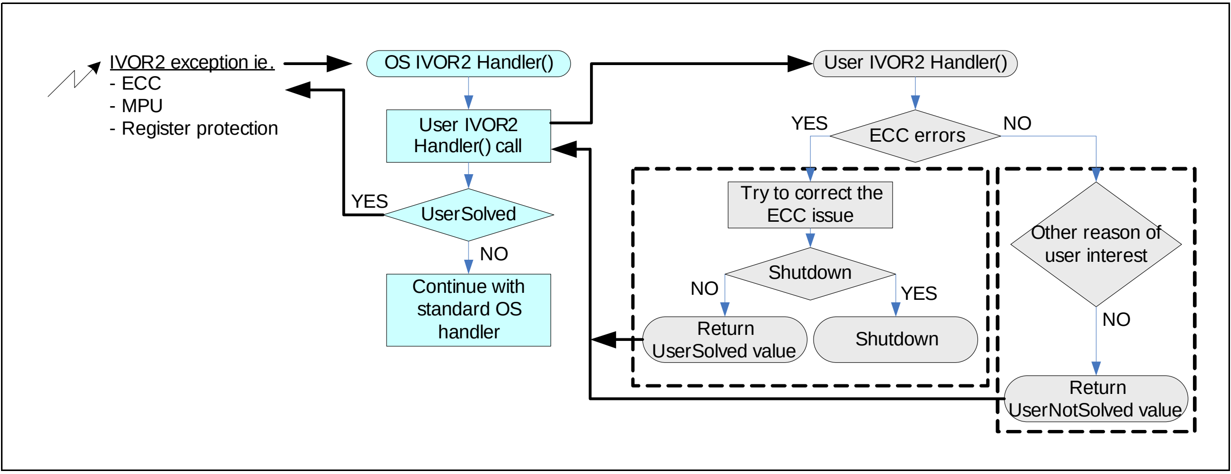
Important thing is that ECC module does not check memory content automatically, but has to be activated by explicit read accesses. There are several possible approaches like reading the memory content piece by piece from the software function or using DMA channel to scan the memory on background. Method selected depends on system requirements and hardware features on particular SPC560x microcontrollers.

5.2 Operating System (OS) exception handling

Whenever operating system is used it usually contains predefined exception handling implementation for most of exception vectors for given family of microcontrollers. Example can be handlers for external interrupts coming from system timer, system call handler or memory protection handlers if memory protection is supported by the microcontroller.

Care must be taken when microcontroller has more exception events assigned to shared vectors like in case of ECC, register protection and memory protection on SPC560x microcontroller family and not all are served by the OS handler. In such case standard OS handlers should give the user support to provide its own implementation and prioritization for these additional exception causes. ECC errors are of particular importance here. *Figure 8* shows an example of such scheme.

Figure 8. OS exception handling with user hook



5.3 Flash memory drivers

Flash driver can easily find if data read from the memory are correct or there was a non-correctable ECC error by reading Flash module control register MCR. There are two status bits linked to ECC events, MCR.EDC bit signaling single bit correction and MCR.EER bit signaling presence of non-correctable EC error during read operation. It is up to the driver and application how it processes these events.

In any case, low level exception events are enabled and activated whenever there is a non-correctable EC error during read access. It can have impact on processing time needed to read the data from Flash memory by the driver.

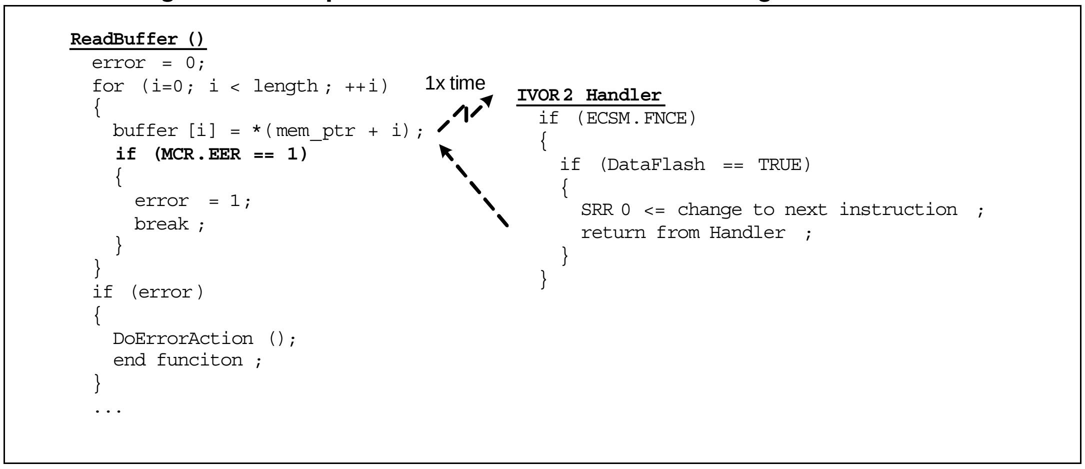
Driver should not rely on ECSM ECC status bits that are used and controlled in low level exception handlers and which are completely independent from the Flash driver.

There are several methods to read multiple data with ECC checking where each method has its pros and cons.

- Check ECC immediately after each read access
- Check ECC after whole data buffer is read
- Low level exception handler linked to the driver ECC check

5.3.1 Check ECC immediately after each read access

Figure 9. Example of Immediate ECC check during read function



```
ReadBuffer ()
  error = 0;
  for (i=0; i < length; ++i)
  {
    buffer [i] = *(mem_ptr + i);
    if (MCR.EER == 1)
    {
      error = 1;
      break ;
    }
  }
  if (error)
  {
    DoErrorAction ();
    end funciton ;
  }
  ...

IVOR2 Handler
  if (ECSM.FNCE)
  {
    if (DataFlash == TRUE)
    {
      SRR0 <= change to next instruction ;
      return from Handler ;
    }
  }
```

Advantage

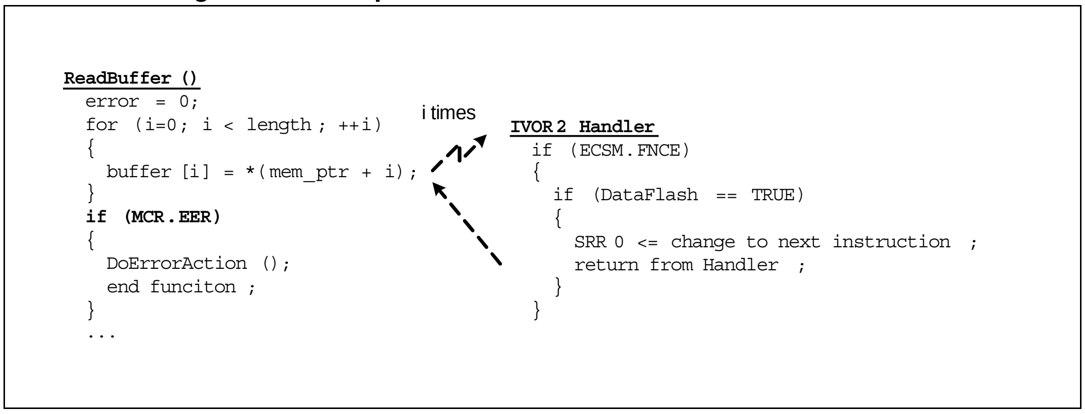
Recognition of corrupted data in the Flash memory is immediate after the first read access. It can help to decrease time in the case where a lot of memory words are corrupted, like power drop during sector erase operation.

Disadvantage

ECC check needs some processing time for each read access. This increases processing time of read operation in general.

5.3.2 Check ECC after whole data buffer is read

Figure 10. Example of ECC check after whole buffer is read



```
ReadBuffer ()
  error = 0;
  for (i=0; i < length; ++i)
  {
    buffer [i] = *(mem_ptr + i);
  }
  if (MCR.EER)
  {
    DoErrorAction ();
    end funciton ;
  }
  ...

IVOR2 Handler
  if (ECSM.FNCE)
  {
    if (DataFlash == TRUE)
    {
      SRR0 <= change to next instruction ;
      return from Handler ;
    }
  }
```

Advantage

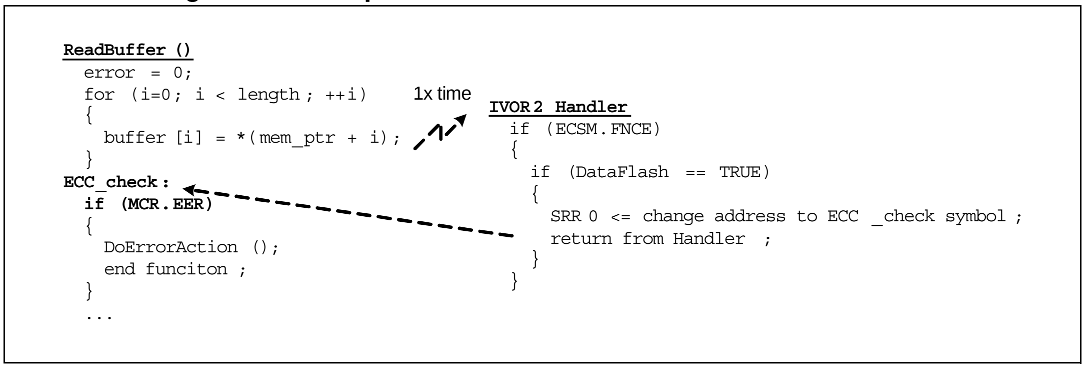
Standard read buffer operation is not delayed with ECC check after each read access. This gives the standard performance in case of good memory sector.

Disadvantage

Any read access with ECC error generates low level exception that runs on background so many times as there are items in the loop. It can have impact on time needed for initial read of a sector with many corrupted cells.

5.3.3 Low level exception handler link to the driver ECC check

Figure 11. Example of Low level link to the driver ECC check



```
ReadBuffer ()
  error = 0;
  for (i=0; i < length; ++i)
  {
    buffer [i] = *(mem_ptr + i);
  }
ECC_check:
  if (MCR.EER)
  {
    DoErrorAction ();
    end funciton ;
  }
  ...
```

```
IVOR2 Handler
  if (ECSM.FNCE)
  {
    if (DataFlash == TRUE)
    {
      SRR0 <= change address to ECC _check symbol ;
      return from Handler ;
    }
  }
```

Advantage

In case of an ECC error during read access, code execution of Read function immediately moves to ECC check part, which should find that the data are corrupted. There is the same performance during errorless operation as well as with ECC errors.

Disadvantage

Because of using exported symbol in the low level exception handler, this symbol should be unique in the whole application. It means that there is only one place in the application and the driver that read from the Flash memory.

Appendix A Document references

1. SPC560D30L1, SPC560D30L3, SPC560D40L1, SPC560D40L3 32-bit MCU family built on the embedded Power Architecture® (RM0045, DocID16886)
2. SPC560B40x, SPC560B50x, SPC560C40x, SPC560C50x 32-bit MCU family built on the embedded Power Architecture® (RM0017, DocID14629)
3. Support microcontrollers SPC560B54x, SPC560B60x and SPC560B64x (RM0037, DocID15700)
4. SPC564Bxx, SPC56ECxx 32-bit MCU family built on the embedded Power Architecture® (RM0070, DocID18196)
5. SPC560P34/SPC560P40 32-bit MCU family built on the embedded Power Architecture® (RM0046, DocID16912)
6. 32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications (RM0022, DocID14891)
7. 32-bit MCU family built on the Power Architecture® embedded category for automotive chassis and safety electronics applications (RM0083, DocID018714)
8. Error Detecting and Error Correcting Codes – Hamming, The Bell Technical Journal 1950

Revision history

Table 6. Document revision history

Date	Revision	Changes
21-Mar-2013	1	Initial release.
18-Sep-2013	2	Updated Disclaimer.