AN5173

Application note

Notes on EDC after ECC functionality

Introduction

This application note describes what the possible causes of the FCCU fault channel EDC after ECC for Flash array are, and how the application can identify them.

This document applies to all microcontrollers belonging to the SPC58 family.

The details described in this document are valid for the ECC logic that protects the integrity of the Flash memories and does not apply to the volatile memories.

1 ECC logic behavior

The code and data Flash uses two different instances of the ECC logic. The algorithm is the same, but the code Flash uses 128-bit of data and 17-bit of parity; the data Flash 64-bit of data and 15-bit of parity.

If the ECC logic recognizes a fault, the hardware takes different reactions depending on whether the source of the error lies in either the data or code Flash.

Note: The fault can be a SEC, DEC, or TED. Section 3 Additional details of the ECC logic describes the behavior in the case of multibit error.

If the error occurs within the code Flash, the hardware:

- Asserts the respective flag in the MCR register of the Flash
- Reports to the MEMU the details of the fault including the address of the faulty location.

Note: Depending on its configuration, the MEMU reports the event to the FCCU. For example, FCCU[28] indicates an uncorrectable error within the Flash.

On the contrary, if the error occurs within the data Flash the hardware:

- Asserts the respective flag in the MCR register of the Flash
- Suppresses the report to the MEMU and the FCCU (refer to section e2eECC on data Flash accesses of the reference manual for all details on this topic).

2 EDC after ECC behavior and handling

The indication of the EDC after ECC error for Flash array is reported to an FCCU input channel according to Table 1.

Table 1. FCCU input channel for EDC after ECC error for the Flash array

Device	FCCU channel number
SPC582B (chorus 1M)	64
SPC584B (chorus 2M)	64
SPC58xCx (chorus 4M)	64
SPC58xNx/SPC58xEx/SPC58xGX (Bernina (1) - Eiger 6M - chorus 6M)	58 for Flash partition RWR0 64 for Flash partition RWR1
SPC58xHx (chorus 10M (2))	56 for Flash partition RWR0 62 for Flash partition RWR1

1. *For the SPC58xNx device, starting from the cut 1.2 in the event of EDC after ECC errors detected on accesses to data and secure data Flash, generation of the fault is suppressed. The related bit in MCR is not set and the event is not forwarded to the FCCU.*
2. *For the SPC58xHx device, in the event of EDC after ECC errors detected on accesses to data and secure data Flash, generation of the fault and forwarding to the FCCU can be controlled by the UT0.DDEF flag. UT0.DDEF default value is 0 (data memory EEE error reporting to FCCU is enabled).*

The FCCU channel EDC after ECC error for the Flash array indicates that the EDC after ECC mechanism has detected a random failure occurring in the ECC logic during a read request to Flash (either data or code Flash).

Although the above analysis is correct, there is another condition that triggers this FCCU channel. If a multibit ECC error occurs in the Flash array, the EDC after ECC can assert the FCCU channel EDC after ECC for the Flash array. Multibit ECC error could occur in Flash because of an interrupted programming operation or Flash block erase interruption (1.).

In this condition, the assertion of the FCCU channel is a fake EDC after ECC fault indication because the cause is not a failure of the ECC logic. The reason for this behavior lies in the hardware implementation of the ECC logic and the EDC after ECC mechanism (see Section 3 Additional details of the ECC logic).

The actual behavior in case of multibit error depends on the combination of data and parity bits. The most common behavior is that the hardware indicates an "EDC after ECC event (2.)" together with the indication of a correction/detection (3.).

However, there are few cases in which the hardware does not signal any fault, or triggers only the "EDC after ECC event."

Note:
1. *This is a frequent condition for data Flash.*
2. *Assertion of the MCR.EEE flag.*
3. *Assertion of either MCR.SBC, MCR.SBC1 or MCR.EER flag.*

Application should manage data Flash corruption (for example, due to interrupted programming or erase operations) at the boot phase after power-on, before the safety application (that enables the FCCU channel EDC after ECC error for the Flash array) starts. It avoids the unwanted / false reaction of the FCCU in this scenario.

Also, for the asynchronous functional resets, where the FCCU is not reset, the application could reconfigure or reset the reaction to the FCCU channel EDC after ECC error for the Flash array before any read operation to the data or secure data Flash and before the safety application starts.

When the FCCU channel EDC after ECC error for the Flash array is asserted, a correct read operation from the Flash array (with no cache or prefetch buffer enabled) in the same partition that generated the error must be done before clearing the FCCU channel, example code is provided in Appendix B Reference code.

If the user disables the FCCU channel EDC after ECC error for both data and code sectors, they must tailor the safety analysis (including the FMEDA) accordingly.

3 Additional details of the ECC logic

The code and data Flash uses two different instances of the ECC logic. All concepts described in this section - however - apply on both of them.

Figure 1 shows the high-level schema of the ECC architecture (1.):

- The blue block represents the ECC logic
- The green blocks represent the EDC after ECC that monitors the integrity of the ECC

Note: *1. This schema is a simplification of the actual implementation. It helps to identify the conditions of the assertion of the EEE flag in case of no correction/detection.*

The ECC logic assesses the presence of single, double, or triple-bit error in a word, including data (D2) and parity (P2). If it evaluates the presence of one of them, it asserts either the SBC, SBC1, or EER flag.

The "DEC-ECC encoder" is a subblock of the EDC after ECC. It recreates the parity (P3) starting from D2c.

Then the comparators compare the physical parity and the corrected one. In case of a mismatch, it asserts the EEE event.

In case of no error in the data and parity, and in the ECC logic, the ECC logic, and the EDC after ECC do not signal an error.

In case of multibit errors, the couple D2/P2 can be either a legal or illegal couple. If it is a legal couple, the hardware behaves as "no error." That is, no error indication from the ECC logic, and EDC after ECC.

If the couple is illegal - in most cases - the ECC logic triggers a wrong correction/detection.

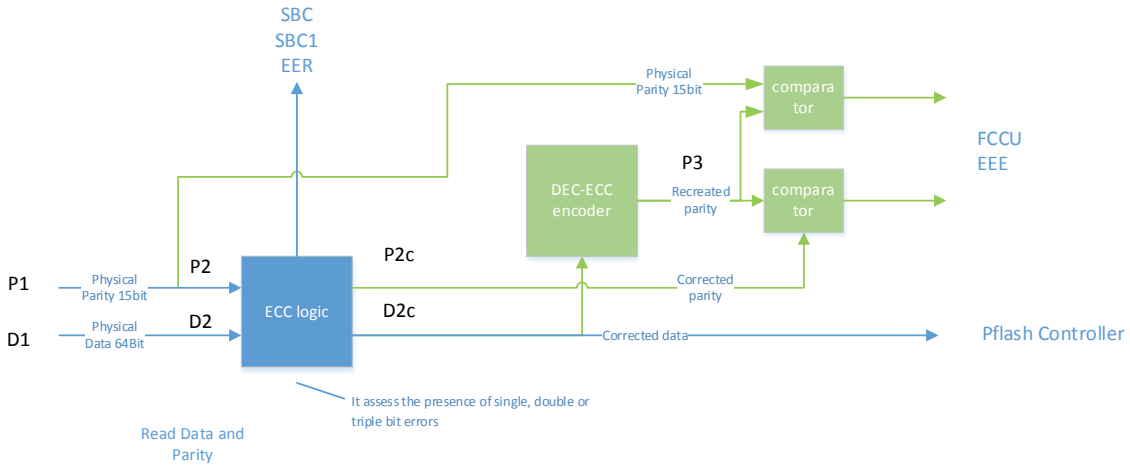
For example, it may correct the data and parity in such a way that:

- P2 = P2c
- D2 <> D2c

As a consequence, the DEC-ECC encoder recreates the parity starting from D2c, and it gets P3 (the legal parity bits corresponding to D2c). Since P3 is different from P2, the comparator triggers an EEE event.

As a result, there is the assertion of the EEE flag together with one of the SBC, SBC1, or ERR flag.

Figure 1. High-level schema of the ECC logic of the Flash (Dx is the data and Px is the parity)



There are few cases - however - in which ECC logic does not detect the presence of a single, double, or triple-bit error, even though the couple P2/D2 is not legal. Consequently:

- P2 = P2c
- D2 = D2c

In this case, the DEC-ECC encoder recreates the parity P3 that is different than P2c (1.). As a result, the comparators assert the EEE event, even if there is not the assertion of SBC, SBC1, or EER flag.

In the data Flash, there are about 32768 combinations (2.) of data and parity bits, around 120 (3.) of them causes this situation.

Note:

1. *P2c is different from P3 because the couple P3/D2c is a valid couple and P2C/D2C is not.*
2. *Around 131072 for the code Flash.*
3. *Around 250 for the code Flash.*

It is important to notice that the purpose of the ECC logic is to assess the presence of single, double, and triple-bit errors. It cannot assess the presence of multibit errors. Other mechanisms - for example, array integrity check and application level checksum - are in place to detect multibit errors.

4 Flash error reporting

This section describes the reporting of the MCR flags in case of different types of error events.

Table 2. MCR flags reporting

Error type	Description
1-bit error	The SBC1 flag indicates this event
2-bit error	The SBC flag indicates this event
3-bit error	The EER flag indicates this event
Error in the ECC logic (wrong correction or wrong detection)	The EEE flag indicates this event when data is legal or containing up to 3-bit errors
More than 3-bit error (that is, >= 4)	In this case, we cannot predict the behavior without knowing the original data and the position of the bit flips. The hardware can react in multiple ways: 1. Do nothing (that is, not signaling of any flag) 2. Assert either SBC, SBC1, or EER without EEE 3. Assert either SBC, SBC1, or EER with EEE 4. Assert EEE only

It is worth highlighting that by design the purpose of the ECC logic is not to detect multibit errors but cope with errors in the data and parity containing maximum 3-bit flips. In this condition, the EDC after ECC monitors the integrity of the ECC logic.

Other mechanisms must detect multibit errors or avoid their occurrence.

For example, running the array integrity check once after the boot before the safety application starts to protect the data with an application level checksum.

An example of a mechanism that decreases the possibility of multibit errors in the array is the multiplexing. The bits of the same word are not one next to the other, but they are spaced of a fixed number of bits.

5 Summary

The hardware asserts the FCCU (EDC after ECC for Flash array) if:

1. A random failure affects the functionality of the ECC logic
2. A multibit error occurs either within the code or data sector

Note: A multibit error means more than three error bits in the same word (that is to say starting from 4 bits).

The application can identify the cause by application level mechanisms that assess the presence of a multibit error.

Depending on the cause and location of the error, the application can jump to the safe state, or try recovering to a valid system state. Clearing of the FCCU channel is possible only after a read from a Flash location in the same partition not affected by the error.

Appendix A Reference documents

Table 3. Reference documents

Document	References
SPC58xNx 32-bit Power Architecture microcontroller for automotive ASILD applications	RM0421, DocID028528
SPC58 2B Line - 32 bit Power Architecture automotive MCU z2 core 80 MHz, 1 MByte Flash, ASIL-B	RM0403
SPC58 4B Line - 32 bit Power Architecture automotive MCU z4 core 120 MHz, 2 MBytes Flash, HSM, ASIL-B	RM0449
SPC58 C Line - 32 bit Power Architecture automotive MCU Dual z4 cores 180 MHz, 4 MBytes Flash, HSM, ASIL-B	RM0407
SPC58 E/G Line - 32 bit Power Architecture automotive MCU Triple z4 cores 180 MHz, 6 MBytes Flash, HSM, ASIL-D	RM0391
SPC58 H Line - 32 bit Power Architecture automotive MCU Triple z4 cores 200 MHz, 10 MBytes Flash, HSM, ASIL-D	RM0452

Appendix B Reference code

```
void FCCU_ALARM_ISR(void) {
    // Call to this function will update the FCCU RF_Sn and Status registers
    Fccu_refresh_status(&FCCU_Status);

    // if FCCU input channel EDC after ECC (that is [64] in this example) is active, clear it
    if (FCCU.RF_S[2].R & 0x1) {
       // Clear EEE bit, it is not necessary for clearing the FCCU input channel
        FLASH_0.MCR.R = 0x10000000;

        // For cases where all caches are enabled, it is necessary
        // to invalidate caches and Flash Line Read Buffers
        // to make sure the Flash array is really accessed

        // Enable after disable Line Read Buffers to invalidate their contents
        PFLASH_0.PFCR1.B.P0_BFEN = 0;
        PFLASH_0.PFCR2.B.P1_BFEN = 0;
        PFLASH_1.PFCR1.B.P0_BFEN = 0;
        PFLASH_1.PFCR2.B.P1_BFEN = 0;
        PFLASH_0.PFCR1.B.P0_BFEN = 1;
        PFLASH_0.PFCR2.B.P1_BFEN = 1;
        PFLASH_1.PFCR1.B.P0_BFEN = 1;
        PFLASH_1.PFCR2.B.P1_BFEN = 1;

        //Invalidate ICache. This is enough if the code is executed from the same partition
of the FLASH that reported the EDC_after_ECC error
        MTSPR(SPR_L1CSR1,0x2);                //Invalidate all, setting the bit in the
special register L1CSR1
        while ((MFSPR(SPR_L1CSR1) & 0x2));    //Wait for completion

        //Invalidate DCache
        MTSPR(SPR_L1CSR0,0x2);                        //Invalidate all, setting the bit in
the special register L1CSR0

        while ((MFSPR(SPR_L1CSR0)&0x2));              //Wait for completion

        // Perform a known good read (e.g. RCHW), if Memory Protection Unit allows it
        dummyReadData = *((uint32_t *)0xFC0000);
   // Clear the FCCU input channel EDC after ECC for flash array ([64] in this example)
        FCCU_Clear_Status(FCCU_CLEAR_CHANNEL(64));
     // Call to this function will update the FCCU RF_Sn and Status registers
        Fccu_refresh_status(&FCCU_Status);

        if (FCCU.RF_S[2].R & 0x1) {
            //for (;;);   // Failed to clear FCCU flag
            // or endless ISR loop if commented out
        }
    } else {
        for(;;);    // Unexpected failure / other FCCU fault
    }
}
```

Revision history

Table 4. Document revision history

Date	Revision	Changes
24-May-2018	1	Initial release.
03-Dec-2021	2	Updated Section Introduction, Section 2 EDC after ECC behavior and handling, Figure 1. High-level schema of the ECC logic of the Flash (Dx is the data and Px is the parity), Section 5 Summary and Section Appendix A Reference documents. Added Section Appendix B Reference code.

Contents

List of tables

List of figures