AN13179

Error Correction Codes Implemented on MPC5744P

Rev. 0 — 3 May 2021 Application Note

Contents

1 Introduction

The Error Correction Code (ECC) is commonly utilized with memories in applications where data corruption via soft errors (SEU) is not easily tolerated. Soft errors can be caused by radiation, electro-magnetic interference, or electrical noise.

The intention of this application note is to describe how the ECC protection is implemented with the MPC5744P device and to understand the MCU's ECC event response.

Because MPC5744P implements the Error Detection Code (EDC) in certain places, this document marginally mentions this topic as well.

This document also slightly compares the approaches used across the MPC57xx family.

NOTE

All diagrams are simplified and they may not contain all implementation details. They may omit the technical details not important for the purpose of this document.

NOTE

The provided information is related to MPC5744P if not stated otherwise.

2 ECC protected memory initialization

2.1 SRAM initialization after power-on-reset

The reset state of the internal SRAM is random so the data and checkbits may contain any data. Most probably, the first read attempt to any address generates a non-correctable ECC error. The SRAM must be initialized after a powerup. It means that the whole SRAM is deleted or written by any value. A 64-bit write is needed to completely define the ECC code for the data unit. Smaller write accesses (32-bit/16-bit/8-bit) cause the read-modify-write operation with ECC error-affected data.

```
# Store number of 128Byte (32GPRs) segments in Counter
e_lis   r5, _SRAM_SIZE@h   # Initialize r5 to size of SRAM (Bytes)
e_or2i  r5, _SRAM_SIZE@l
e_srwi  r5, r5, 0x7        # Divide SRAM size by 128
mtctr   r5                 # Move to counter for use with "bdnz"
# Base Address of the internal SRAM
e_lis   r5, _SRAM_BASE_ADDR@h
e_or2i  r5, _SRAM_BASE_ADDR@l
# Fill SRAM with writes of 32GPRs
sram_loop:
e_stmw  r0,0(r5)            # Write all 32 registers to SRAM
```

```
e_addi  r5,r5,128           # Increment the RAM pointer to next 128bytes
e_bdnz  sram_loop           # Loop for all of SRAM
```

2.2 SRAM initialization after functional reset

You can omit the SRAM initialization for the functional reset sources. When a functional reset event occurs, a partial reset sequence is applied to the chip starting from PHASE1[FUNC], keeping the system memory content preserved.

2.3 Initialization of other embedded SRAM memories

2.3.1 eDMA RAM arrays

Automatically:

The initialization of the eDMA Transfer Control Descriptor (TCD) memory is performed by the eDMA controller itself after a reset.

The initialization runs for 256 eDMA clock cycles. All fields in the TCD memory are initialized to 0. All application read or write accesses to the TCD are delayed until the initialization is finished.

2.3.2 FlexCAN RAM arrays

Manually:

The whole FlexCAN memory must be initialized before starting its operation to have the checksum bits in the memory properly updated. The WRMFRZ bit in the Control 2 Register (CTRL2) grants write access to all memory positions from 0x080 to 0xADF.

2.3.3 FlexRay RAM arrays

Automatically:

The initialization of the CHI LRAM is performed by the CC when it leaves the Disabled Mode. The unitization runs for 45 CHI clock cycles. All fields in the FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBDORn, and LEETRn registers are initialized to 0. All application read or write accesses to these registers are delayed until the initialization is finished.

3 Used error detection/correction codes

3.1 Terminology used within this document

- Error Correction Code (ECC) - adding checksum bits to protected data. All employed ECC algorithms provide the SECDED capability.
- Single Error Correction/Double Error Detection (SECDED).
- 1-bit/single-bit/correctable ECC error - 1 faulty bit within the data unit or checksum that can be corrected by SECDED.
- 2-bit/multi-bit/non-correctable/uncorrectable ECC error – 2 (or more) faulty bits within the data unit or checksum that can only be detected by SECDED.
- End-to-End ECC (e2eECC) - provides an additional layer of safety by including the ECC on all bus transactions. The ECC for the transfer is generated on the transmitting end of the transaction and checked at the receiving end.
- Error Detection Code (EDC) - adding checksum bits to the protected data. The employed EDC algorithms provide Double Error Detection capability. The term EDC is used in our documents in multiple meanings. In most cases, it is additional EDC protection (supervision), checksum being added and subsequently removed from/to data, address, attribute, or other signals, according the specific needs of the protected transfer or memory module. It is further protection capable to find an ECC malfunction.

- ECC transformation (Checkbit Transformation) - internal busses may use different granularities (either 64-bit or 32-bit). On the interface of two different busses, the checkbit transformation (i.e. logical operations changing data checksum from one format to another) or the reverse transformation is necessary.
- ECC manipulation - further generalization of any ECC checkbits re-coding, either because of a less than 64-bit access or due to removing the address portion from the ECC (changing from the e2eECC on internal busses to the ECC in the target memory).

3.2 Used ECC algorithms and error responses

The employed ECC protection variants used with the MPC57xx internal memories use the Hsiao Code in 64-bit or 32-bit granularities.

3.3 e2eECC protection of transfers over system buses

MPC5744P (e200z4251n3):

Data checkbit generation - external interfaces, 64-bit data ECC granularity

Internal data checkbit generation - 32-bit data ECC granularity

Address portion of checkbit generation, 64- or 32-bit data ECC granularity

ICACHE and IMEM data checkbit generation - 64-bit ECC granularity

DCACHE and DMEM internal data checkbit generation - 32-bit data ECC granularity

3.4 Internal flash

MPC5744P: 64-bit data + 8-bit ECC

3.4.1 Code flash

Single-bit ECC events on code flash accesses are automatically corrected and reported to the MEMU (if enabled in the flash controller by the procedure described in Intentional generation of FLASH 1b ECC error). On a multi-bit ECC event, the core responds with a bus error, as summarized in e200z2/z4/z7 core response on ECC event, and an error is reported to the MEMU module.

3.4.2 Data flash

The reporting of ECC events on data flash accesses is device-specific. It can be configurable or suppressed, as shown in Table 1.

When the ECC event reporting is suppressed, single-bit and multi-bit ECC errors are not reported (either to the core or to the MEMU module). On a multi-bit ECC event, the corrupted read data is replaced with a fixed, ECC-clean illegal opcode value.

When the ECC event reporting is configurable and enabled, single-bit ECC events on data flash accesses are automatically corrected and reported to the MEMU (if enabled in the flash controller by the procedure described in Intentional generation of FLASH 1b ECC error). On a multi-bit ECC event, the core responds with a bus error, as summarized in e200z2/z4/z7 core response on ECC event, and an error is reported to the MEMU module.

When the ECC event reporting is configurable and disabled, it behaves the same way as if it was permanently suppressed.

Table 1. ECC on data flash accesses

	ECC event reporting on data flash access is suppressed permanently/optionally	Returned ECC-clean illegal opcode value
MPC5744P	permanently	0xFFFF_FFFF
MPC5746C	permanently (only HSM data flash)	0x1555_1555

Table continues on the next page...

Table 1. ECC on data flash accesses (continued)

	ECC event reporting on data flash access is suppressed permanently/optionally	Returned ECC-clean illegal opcode value
MPC5748G	permanently (only HSM data flash)	0x1555_1555
MPC5746R	optionally, PFCR3[DERR_SUP]	0x1555_1555
MPC5775K	permanently	0x1555_1555
MPC5777C	optionally, PFCR3[DERR_SUP]	0x1555_1555
MPC5777M	permanently	0x1555_1555

3.5 Internal SRAM

MPC5744P: 64-bit data + 8-bit ECC

Single-bit ECC events on the internal SRAM accesses are automatically corrected and reported to the MEMU. On a multi-bit ECC event, the core responds with a bus error, as summarized in e200z2/z4/z7 core response on ECC event, and an error is reported to the MEMU module.

3.6 ECC on other embedded internal SRAM memories

For a complete picture, the following MPC5744P's embedded RAMs are ECC-protected as well:

- D-MEM
- I-CACHE and D-CACHE
- FlexRay RAM arrays
- FlexCAN RAM arrays
- eDMA RAM arrays

3.7 Used EDC protection (supervision)

3.7.1 Flash/SRAM

Additional EDC transfer protection may be located in the flash array or flash controller due to ECC manipulation (either because of less than 64-bit access ECC transformation or because of the transfer protection between the flash array and the flash controller when the ECC is re-coded due to a removal of the address portion from the transferred packet address, data, or e2eECC).

3.7.2 Crossbar Integrity Checker (XBIC)

The XBIC is sub-module verifying the integrity of the attribute information for XBAR transfers using an 8-bit Error Detection Code (EDC). The XBIC integrity checking is independent from the end-to-end ECC that covers the transfer address and data.

The EDC(72,64) code, which protects against single- and double-bit error of 64-bit attribute information (transfer direction, type, size, protection control, burst type, and so on) is used.

4 MPC5744P ECC/EDC system implementation

The system ECC/EDC principles of transfer protection by error detecting/correcting code are shown in Figure 1 and Figure 2. Two use cases are shown: core access to/from the internal flash and SRAM. The diagrams show the used checksum on the transfer

paths (from the XBAR master over the system busses to the XBAR slave). They also show the error-reporting paths and relations between the error collecting and managing units (MEMU, XBIC) and the reaction modules (FCCU, RGM, and INTC).

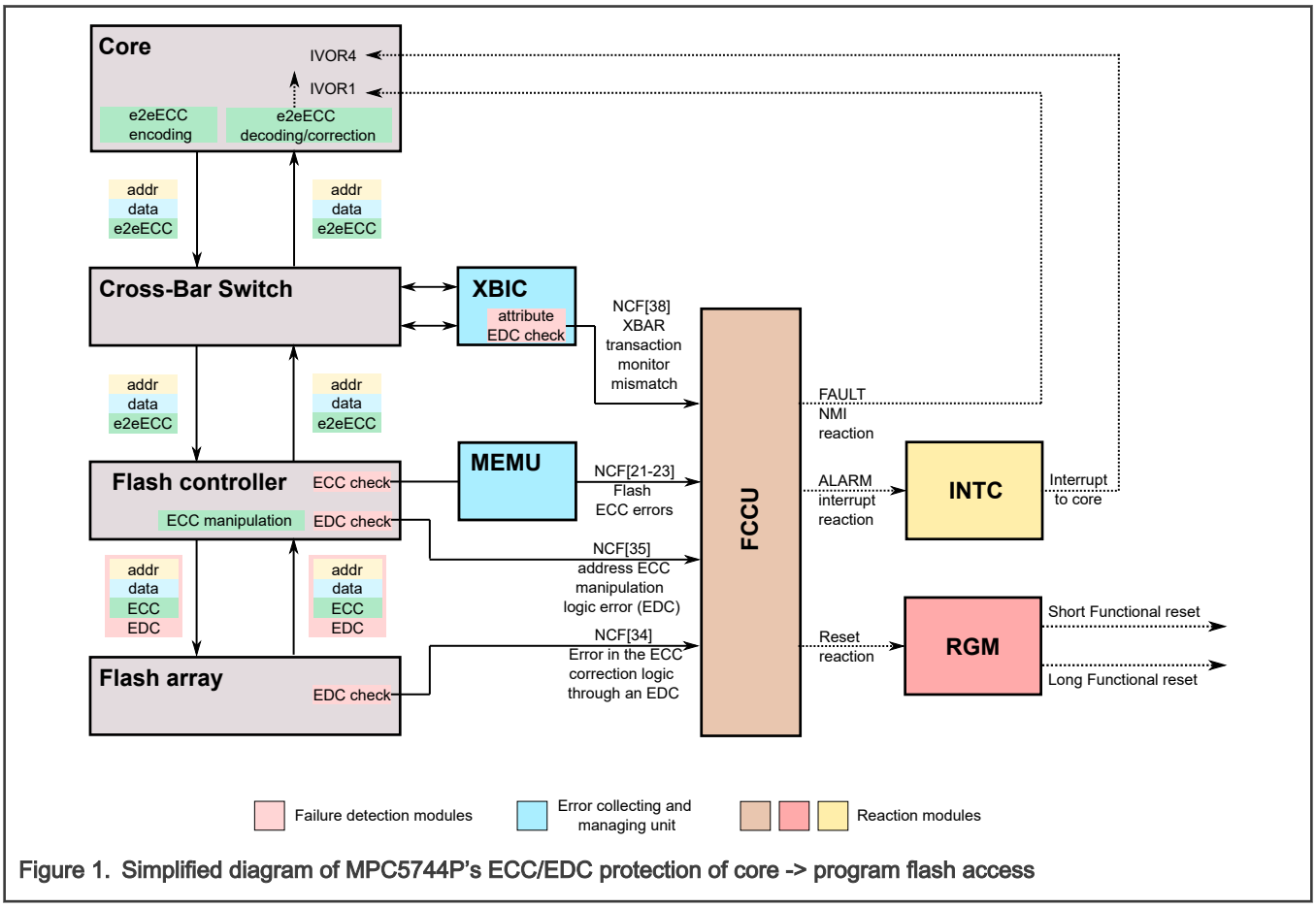

Figure 1. Simplified diagram of MPC5744P's ECC/EDC protection of core -> program flash access

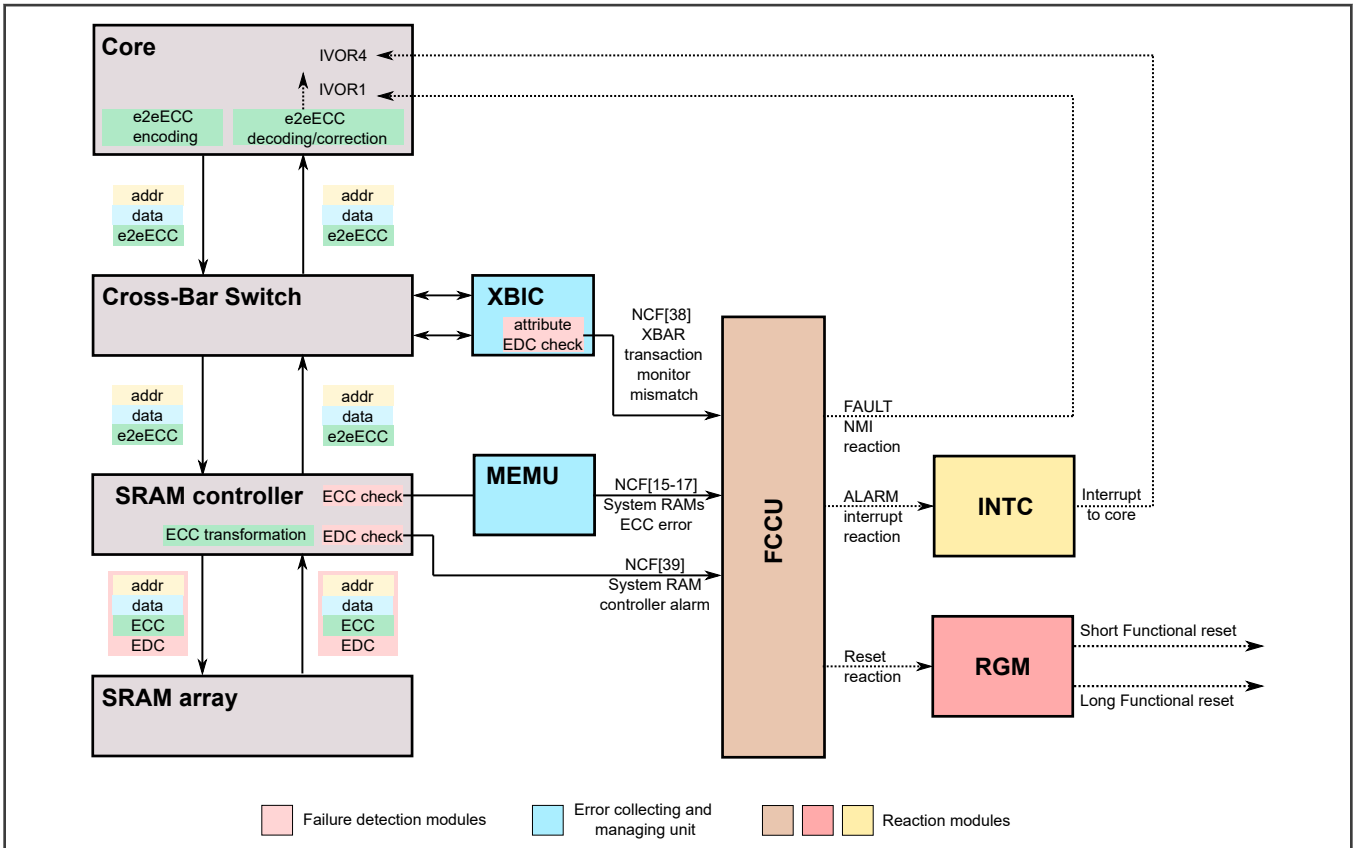

Figure 2. Simplified diagram of MPC5744P's ECC/EDC protection of core -> SRAM access

5 e200z2/z4/z7 core response on ECC event

The correctable (single-bit) errors are automatically corrected and they can be tracked by the MEMU.

The non-correctable (multi-bit) ECC error causes the Machine Check Exception (IVOR1) and a potential interrupt generated via the MEMU-related FCCU non-critical fault input and its ALARM state (IVOR4 for the software vector mode or vector 488 for the hardware vector mode).

Table 2. Core reaction to detection of multiple-bit ECC error for e200z2/z4/z7

MSR[EE]	MSR[ME]	Access type	Result
x	0	Instruction or data	Machine Check Exception (IVOR 1) Error flags in the MCSR register are ignored
x	1	Instruction or data	Machine Check Exception (IVOR 1) Error flags in the MCSR register must be cleared in the exception service routine to avoid IVOR 1 recall

The MEMU/FCCU configuration is further described in Interrupt handling of non-correctable ECC error.

6 Behavior in case EDC event occurs

The MPC57xx architecture offers advanced EDC transfer protection, such as the XBIC attribute EDC, RAM controller EDC, and flash controller EDC (see the device reference manual for a specific implementation).

The detection of such errors triggers the EDC-related FCCU non-critical fault input. The FCCU configuration is further described in FCCU.

7 Correctable ECC error servicing

The correctable ECC error servicing is not needed, because 1-bit ECC errors are automatically corrected during data/instruction read. The detection of 1-bit errors could sense gradual degradation of the flash memory content caused by aging. A 1-bit error may be corrected by reading the data and writing them back.

8 Non-correctable ECC error servicing

The fixing of multi-bit ECC errors is application-dependent and it must be part of the ECC error interrupt handling.

It can be based on the IVOR1 exception handler or IVOR4 interrupt handler, as described in IVOR1 exception handling of non-correctable ECC error and Interrupt handling of non-correctable ECC error.

8.1 IVOR1 exception handling of non-correctable ECC error

This document describes the IVOR1 (machine check) handling, because it is the most common one and it is compatible with all e200 cores. The precondition is to have MSR[ME]=1, because it guarantees that the exception is not only directly associated with the current instruction execution stream (synchronous exceptions), but also with those reported by the subsystem as a bus error termination (asynchronous exceptions, for example cache line filling). However, this approach still catches only errors related to the core.

If ECC non-correctable errors caused by other master than the core are supposed to be serviced, use the handling based on the MEMU interrupt (Interrupt handling of non-correctable ECC error) or a combination of both approaches.

8.1.1 Machine Check Syndrome Register (MCSR)

The register provides a possibility to differentiate between the sources of machine check exceptions. The exception service routine should analyze the root cause of the exception:

- The error is caused by reading ECC-corrupted data sets MCSR[MAV, LD, BUS_DRERR].
- The error is caused by writing to the area affected by the ECC multibit error sets MCSR[MAV, LD, BUS_DRERR, BUS_WRERR]. This can be achieved by 32-bit, 16-bit, and 8-bit writes, because it behaves as a read-modify-write operation above 64bits.
- The error is caused by an attempt to execute an instruction affected by the ECC multibit error sets MCSR[MAV, IF, BUS_IRERR].
- The error is caused by the cache-line-filling set MCSR[MAV, BUS_DRERR] or MCSR[MAV, BUS_IRERR] when there are data affected by the ECC multibit error within this line, but not in the currently loaded data.

MCSR[MAV] - indicates that the address in the MCAR was updated by hardware. Note that the next update is only performed when this bit is explicitly cleared by the W1C operation (Write 1 to Clear).

8.1.2 Machine Check Address Register (MCAR)

This register contains the effective (when MCSR[MEA]=1) or physical (when MCSR[MEA]=0) addresses, for which the asynchronous type of the machine check exception was raised (not all machine sources update this register).

The address is valid only when MCSR[MAV] was cleared before the exception.

8.1.3 Machine Check Save/Restore Register 0 (MCSRR0)

This register contains the address of the instruction that caused the exception. At the end of the exception service routine, the “rfmci” instruction loads the content of this register as the return address (program counter).

To return to the program flow before the exception occurred (i.e. after the instruction that caused the exception), increase the exception returning address by the length of instruction causing an exception (in case of BookE by 4, in case of VLE by 2 or 4, according to the instruction opcode).

For details, see the *VLE 16-bit and 32-bit Instruction Length Decode Algorithm* (document AN4648).

Table 3. Read content of address given by MCSRR0 register during machine check exception

Bit3	Bit0	Instruction was	Increment MCSRR0 by
0	0	16-bit	2
0	1	16-bit	2
1	0	32-bit	4
1	1	16-bit	2

8.2 Interrupt handling of non-correctable ECC error

According to the implementation, the MPC57xx devices have either the MEMU or ERM modules.

Table 4. Reporting module used

	Memory Error Management Unit (MEMU)	Error Reporting Module (ERM)
MPC5744P	Yes	No
MPC5746C	Yes	No
MPC5748G	Yes	No
MPC5746R	Yes	No
MPC5775K	Yes	No
MPC5777C	No	Yes
MPC5777M	Yes	No

The following two sections briefly describe the MPC5744P's MEMU and FCCU modules, because it is necessary to understand their functionality to create an application-specific ECC/EDC handler.

8.2.1 MEMU

The Memory Error Management Unit (MEMU) is a module dedicated for collection and reporting of error events associated with the ECC. It provides a set of registers offering extended information about detected ECC events. In comparison to the approach described in IVOR1 exception handling of non-correctable ECC error, MEMU has the following benefits:

- It aggregates information from several sources and it is capable to distinguish between three instances:
  - System RAM ECC event.
  - Peripheral RAM ECC event.
  - Internal FLASH ECC event.
- It allows the detection of ECC events from XBAR masters other than the core (eDMA).

- It can detect single-bit correctable ECC events.
- It has a reporting table to catch multiple ECC error events:
  - 10 entries for the System RAM single-bit ECC event.
  - 2 entries for the Peripheral RAM single-bit ECC event.
  - 20 entries for the Internal FLASH single-bit ECC event.
  - 1 entry for the System RAM multi-bit ECC event.
  - 1 entry for the Peripheral RAM multi-bit ECC event.
  - 1 entry for the Internal FLASH multi-bit ECC event.
- Single-bit correctable and multi-bit uncorrectable ECC events are reported to the FCCU with a dedicated fault for every instance.
- If the reporting table entries are full and a new unique error is reported by the system, the "ECC error overflow" signal can be indicated to the FCCU. If it cannot process all simultaneously-arriving reports (because it aggregates address information from several sources), the "buffer overflow" signal is signaled to the FCCU with a dedicated fault for every instance. Figure 3, Figure 4, and Figure 5 show the ECC error-reporting paths.

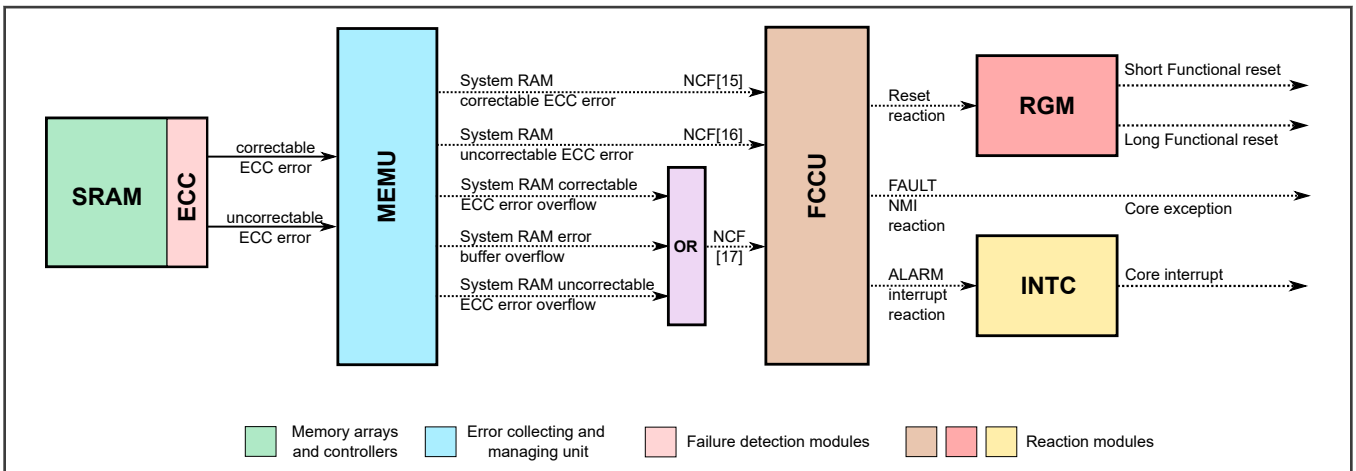

Figure 3. MPC5744P's SRAM ECC error reporting path

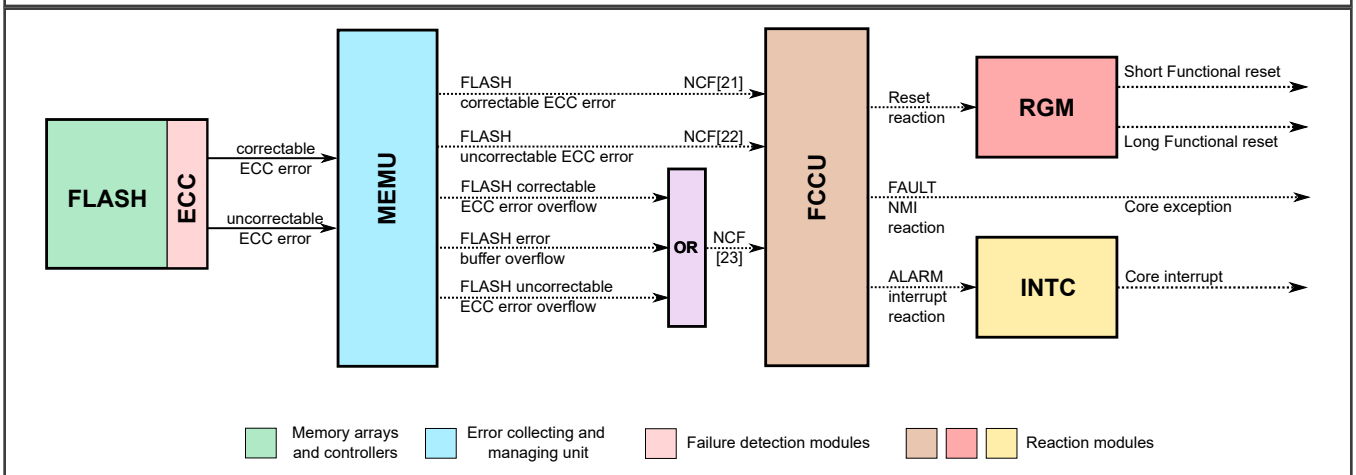

Figure 4. MPC5744P's FLASH ECC error reporting path

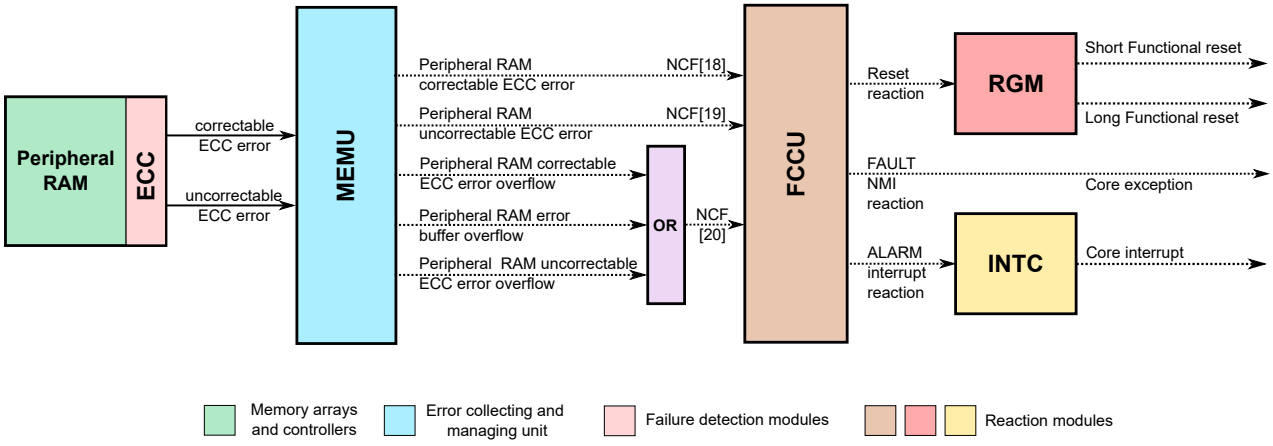

Figure 5. MPC5744P's Peripheral RAM ECC error reporting path

Table 5. MEMU module registers

Register description	Register
MEMU_CTRL	Control register
MEMU_ERR_FLAG	Error flag register
MEMU_DEBUG	Debug register (forcing FCCU faults)
MEMU_SYS_RAM_CERR_STSn MEMU_SYS_RAM_CERR_ADDRn MEMU_SYS_RAM_UNCERR_STS MEMU_SYS_RAM_UNCERR_ADDR MEMU_SYS_RAM_OFLWn	System RAM reporting registers
MEMU_PERIPH_RAM_CERR_STSn MEMU_PERIPH_RAM_CERR_ADDRn MEMU_PERIPH_RAM_UNCERR_STS MEMU_PERIPH_RAM_UNCERR_ADDR MEMU_PERIPH_RAM_OFLWn	Peripheral RAM reporting registers
MEMU_FLASH_CERR_STSn MEMU_FLASH_CERR_ADDRn MEMU_FLASH_UNCERR_STS MEMU_FLASH_UNCERR_ADDR MEMU_FLASH_OFLWn	Internal FLASH reporting registers

Table 6. Used abbreviations in MEMU_ERR_FLAG register

Instance	ECC error type	Overflow type
PR => Peripheral RAM F => Flash SR => System RAM and MBIST	CE => ECC Correctable Error (single-bit) UCE => ECC Uncorrectable Error (multi-bit)	CEO => ECC Correctable error Overflow (single-bit) UCO => ECC Uncorrectable error Overflow (multi-bit) EBO => ECC Error Buffer (concurrent) Overflow

The MEMU handler is application-specific (and you may see such ones in example codes 1-3). Here are certain points to mention:

- The MEMU module is always active.
- The multi-bit ECC error caught during core access also causes the bus error to the core (IVOR1).
- The MEMU can only trigger the NCF faults which go to the FCCU that must be configured for the interrupt reaction on the NCF faults.
- If an error was injected by the Error Injection Module (EIM), disable the EIM injection at the very start of the ISR.
- The MEMU handler consists of the following parts:
  1. Examine the MEMU_ERR_FLAG to find out the instance, ECC error, or overflow type.
  2. For a given instance and type, perform a scan of all entries, looking for entries with VLD (valid) bit set.
  3. Examine the rest of reporting registers related to a particular valid entry.
  4. Perform the application-specific countermeasures.
  5. Invalidate the entry.
  6. If there is no other error, clear MEMU_ERR_FLAG by W1C. Otherwise, return to point “a”.

8.2.2 FCCU

Table 7 shows a list of ECC/EDC-related FCCU fault sources. Every NCF can be set for the following reactions:

- IRQ (when the FCCU transits from the NORMAL to the ALARM state)
- Short functional reset
- Long functional reset
- NMI (when the FCCU transits trom the ALARM to the FAULT state)

Table 7. FCCU non-critical faults mapping (extract)

Non-critical fault	Source	Signal description
NCF[15]	MEMU	System RAMs correctable ECC error
NCF[16]	MEMU	System RAMs uncorrectable ECC error
NCF[17]	MEMU	System RAMs error overflow (ORing of all overflows)
NCF[18]	MEMU	Peripheral RAMs correctable ECC error
NCF[19]	MEMU	Peripheral RAMs uncorrectable ECC error

Table continues on the next page...

Table 7. FCCU non-critical faults mapping (extract) (continued)

Non-critical fault	Source	Signal description
NCF[20]	MEMU	Peripheral RAMs error overflow (ORing of all overflows)
NCF[21]	MEMU	Flash correctable ECC error
NCF[22]	MEMU	Flash uncorrectable ECC error
NCF[23]	MEMU	Flash error overflow (ORing of all overflows)
NCF[34]	PFLASH	Error in the ECC correction logic through an EDC
NCF[35]	PFLASH	Alarm indicating that the flash memory controller detected an error
NCF[38]	XBAR	XBAR transaction monitor mismatch
NCF[39]	PRAMC	System RAM controller alarm

For a proper FCCU setting, see *Using FCCU on MPC5744P* (document AN5284).

For example, the following FCCU initialization is used for the Example Codes, enabling the ALARM IRQ for all MEMU-related fault sources:

```
void FCCU_Init(void)
{
      /* clear possible faults*/
      ClearNCF();

      /* Unlock configuration */
      FCCU.TRANS_LOCK.R = 0xBC;
      /* provide Config state key */
      FCCU.CTRLK.R = 0x913756AF; //key for OP1
      /* enter config state - OP1 */
      FCCU.CTRL.R = 0x1; //set OP1 - set up FCCU into the CONFIG mode
      /* wait for successful state transition */
      while (FCCU.CTRL.B.OPS != 0x3); //operation status succesful

      /* Non-critical fault enable for all MEMU sources */
      FCCU.NCF_TOE[0].R = 0xFFFFFFFF; //ALARM Timeout Enable
      FCCU.NCF_E[0].R = 0x00FF8000; // NCF[15]-NCF[23]
      FCCU.IRQ_ALARM_EN[0].R = 0x00FF8000;

      /* set up the NORMAL mode of FCCU */
      FCCU.CTRLK.R = 0x825A132B; //key for OP2
      FCCU.CTRL.R = 0x2; //set the OP2 - set up FCCU into the NORMAL mode
      while (FCCU.CTRL.B.OPS != 0x3); //operational status succesful
}

void ClearNCF(void)
{
      uint32_t i,b[4];
      for(i=0;i<4;i++)
      {
            FCCU.NCFK.R = FCCU_NCFK_KEY;
            FCCU.NCF_S[i].R = 0xFFFFFFFF;
            while(FCCU.CTRL.B.OPS != 0x3)
```

```
            {};  /* wait for the completion of the operation */
            b[i]=FCCU.NCF_S[i].R;
      }
}
```

9 ECC error injection methods

9.1 Intentional generation of SRAM 1b/2b ECC error

Table 8. Options for 2b embedded RAM ECC error injection across MPC57xx product line

	Error Injection Module (EIM)	SRAM ECC error injection with E2EECSR (DCR 511)	Indirect Memory Access (IMA)
MPC5744P	for DMA TCD RAM array	Yes	No
MPC5746C	No	Yes	No
MPC5748G	No	Yes	No
MPC5746R	for DMA TCD RAM array	Yes	No
MPC5775K	for DMA TCD RAM array	Yes	No
MPC5777M	No	Yes	Yes
MPC5777C	for DMA TCD RAM array, FEC RAM array, internal SRAM array	No	No

9.1.1 System RAM

For the system RAM, the method with the core register E2EECSR (DCR 511) may be used to invert the selected bits of the originally-calculated e2eECC checkbits during the core write.

If you enable the E2EECSR0[INVC]=1 error injection and set up a proper mask to the E2EECSR0[CHKINVT] bits, a subsequent write to the SRAM creates an error and the following read of this area causes a bus error.

Preconditions use a 64-bit aligned write for the store operation to avoid checkbits transformations to store corrupted data to the target RAM memory.

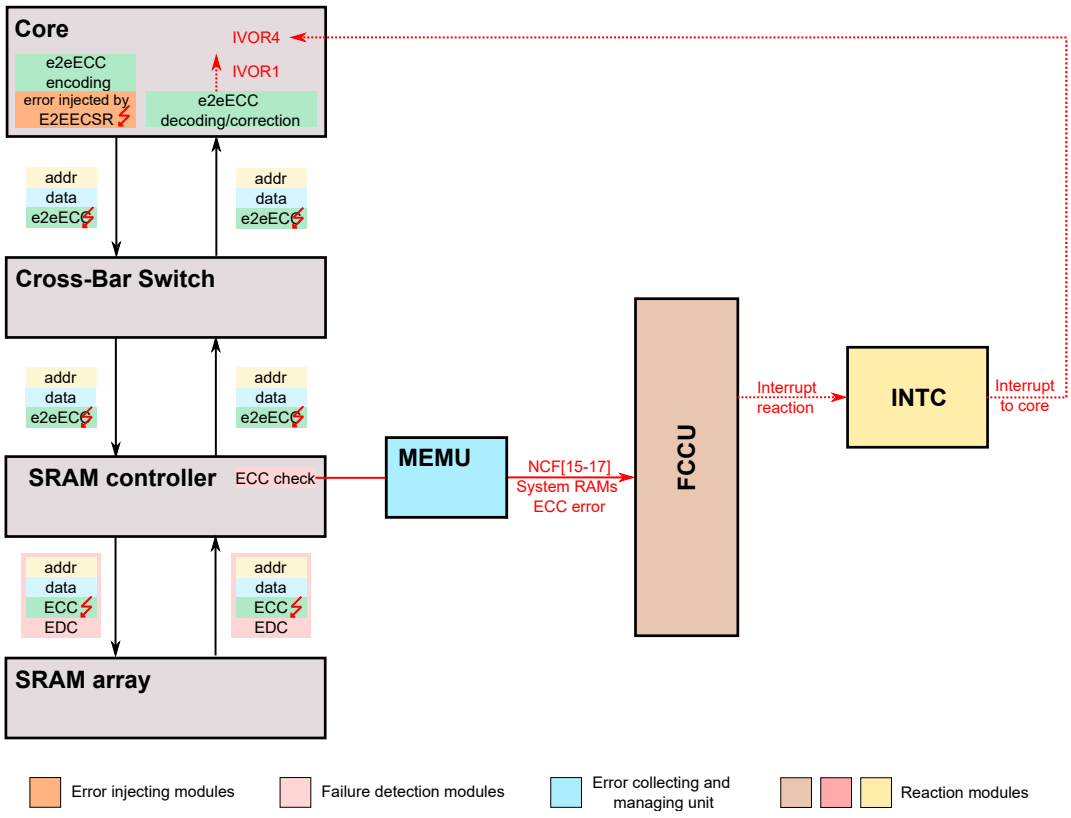

Figure 6. RAM ECC error injection with core register E2EECSR (DCR 511)

9.1.2 eDMA RAM arrays

The ECC error in the DMA RAM memory can be simulated using the Error Injection Module (EIM) on the MPC5744P device. EIM can simulate the single-bit and multi-bit errors by inverting the selected lines on the data/checkbit bus of the DMA RAM memory during a read operation.

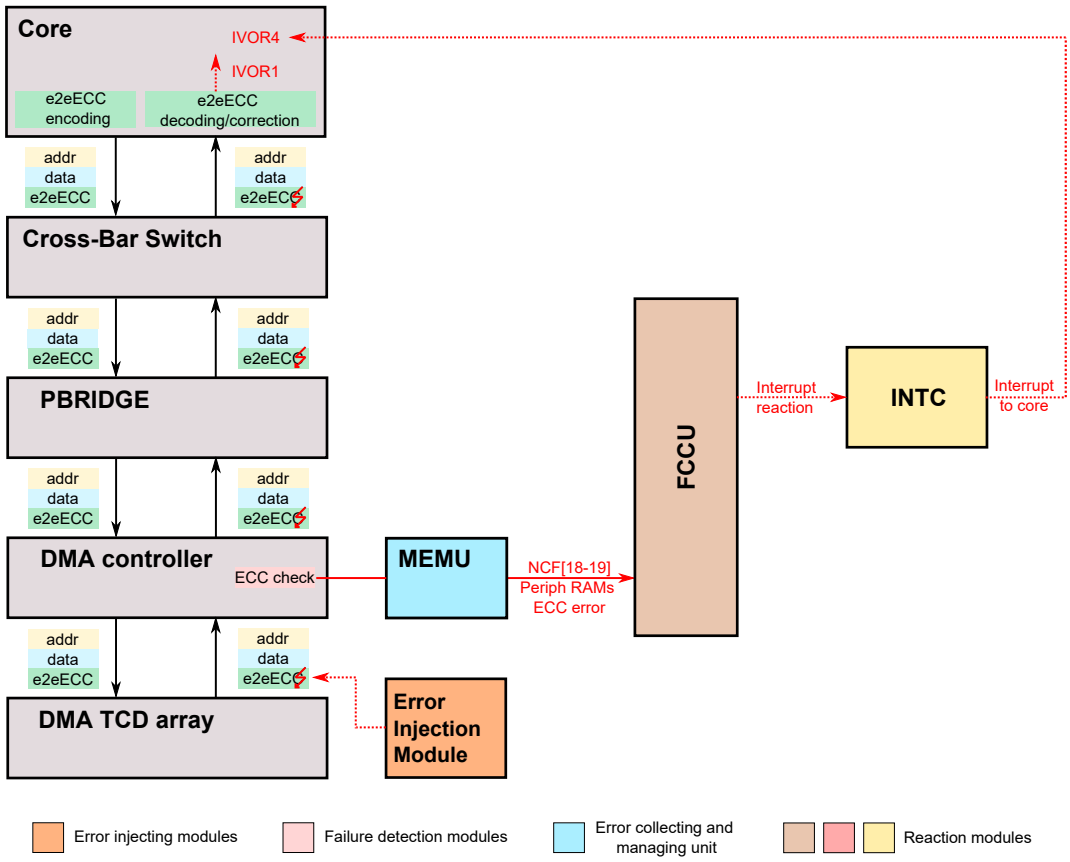

Figure 7. PERRAM ECC error injection using Error Injection Module (EIM)

9.1.3 Other embedded SRAM memories

The injection methods for the rest of internal embedded SRAM memories are as follows:

- D-MEM - Uses DMEMCTL0 (DCR496). If DMEMCTL0[DPEIE] is set, the subsequent write to D-MEM creates a 2b ECC error.
- I-CACHE - If L1CSR1[ICEI] is set, any instruction cache line filled to the instruction cache data has associated the two most significant parity check bits inverted in the instruction cache data array for each doubleword loaded (simulates a 2b ECC error).
- D-CACHE - If L1CSR0[DCEI] is set, any cache line filled to the data cache data array has associated the two most significant parity check bits inverted in the data array for each word loaded (simulates a 2b ECC error). Additionally, inverted parity bits are generated for any data stored into the data cache data array on a store hit (injects a 2b ECC error).
- FlexRay RAM arrays - The error injection mode is configured by the EIM configuration bit in the ECC Error Report and Injection Control Register (FR_EERICR). When the error injection is enabled (FR_EERICR[EIE] = 1), each write access to the configured memory location is distorted (XORed with R_EEIDR[DATA]) or directly written by the data specified by R_EEIDR[DATA], according to the Error Injection Mode (FR_EERICR[EIM]).
- FlexCAN RAM arrays – The Error Injection Address Register (CAN_ERRIAR), Error Injection Data Pattern Register (CAN_ERRIDPR), and Error Injection Parity Pattern Register (CAN_ERRIPPR) are used to inject errors in memory reads to force errors. The injection is done by flipping the data and parity bits correspondent to the bits 1 in ERRIDPR and ERRIPPR. The injection can be selected specifically for memory accesses requested by the host or by FlexCAN internal processes (CAN_MECR[HAERRIE,FAERRIE]). In case of 64-bit accesses, the CAN_MECR[EXTERRIE] bit extends the error injection on 32-bit memory accesses to the complementary 32-bit word using the same 32-bit error injection data and parity words.

9.2 Intentional generation of FLASH 2b ECC error

9.2.1 Option 1 - flash over programming

ECC errors can be generated in the flash by overprogramming memory locations.

A multiple bit error is detected but not corrected. Therefore, it is easier to see when it is injected. A procedure for creating a multiple bit error is as follows:

1. Write the original data A = 0x0045000000000000 to a flash memory location.
2. Overprogram data A to data B = 0x0058000000000000 in the same flash memory location.

This creates a multiple-bit ECC error and it is flagged in the ECC module.

Make sure to choose data patterns that have different ECC checkbits, because over-programming does not necessarily generate an ECC error. For instance, the patterns shown in Table 9 have the same ECC checkbits and they can be overwritten without generating an ECC error (this feature is utilized by EEPROM emulation drivers).

Table 9. Examples of data patterns with same ECC checkbits

Doubleword	Checkbits
0xFFFF_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_FFFF_0000	0xFF
0xFFFF_FFFF_0000_FFFF	0xFF
0xFFFF_0000_FFFF_FFFF	0xFF
0x0000_FFFF_FFFF_FFFF	0xFF
0xFFFF_FFFF_0000_0000	0xFF
0xFFFF_0000_FFFF_0000	0xFF
0x0000_FFFF_FFFF_0000	0xFF
0xFFFF_0000_0000_FFFF	0xFF
0x0000_FFFF_0000_FFFF	0xFF
0x0000_0000_FFFF_FFFF	0xFF
0xFFFF_0000_0000_0000	0xFF
0x0000_FFFF_0000_0000	0xFF
0x0000_0000_0000_0000	0xFF

9.2.2 Option 2 - reading of UTEST area

The MPC57xx device family offers a set of test patterns, pre-programmed in the factory (see Table 10). For a multi-bit error, read address 0x00400060.

Table 10. UTEST flash memory (extract)

Start address	End address	Size (bytes)	Description	Notes
0x00400040	0x0040005F	32	Customer single-bit correction area	*
0x00400060	0x0040007F	32	Customer double-bit detection area	*
0x00400080	0x0040009F	32	Customer EDC after ECC area	*

* Programmed by NXP to include ECC/EDC errors to allow testing of ECC/EDC hardware

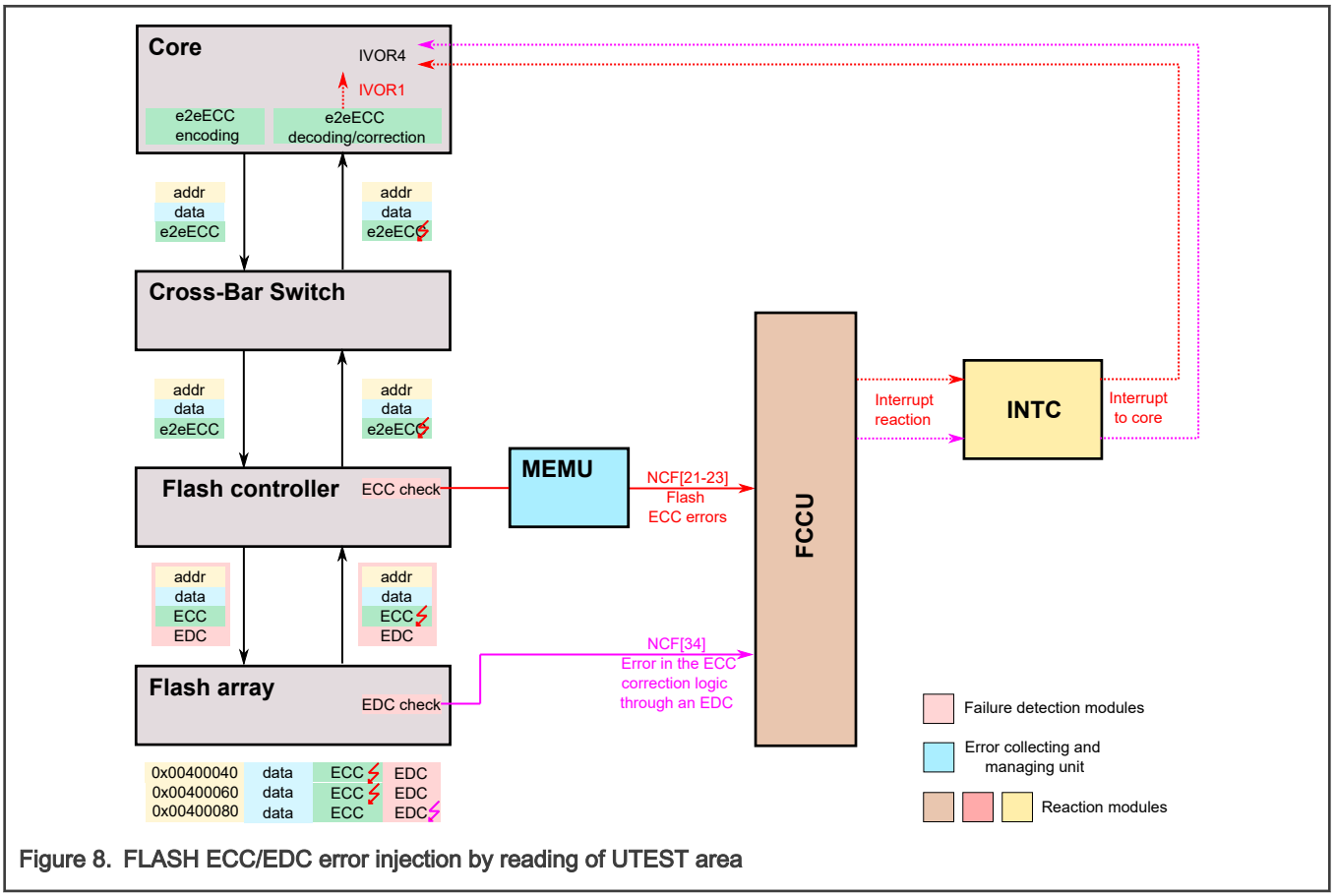

Figure 8. FLASH ECC/EDC error injection by reading of UTEST area

9.3 Intentional generation of FLASH 1b ECC error

The 1b ECC error is corrected without reporting it into the MEMU/ERM module by default (C55FMC_MCR[SBC] is not set on the 1b ECC event). Enable it by configuring the PFLASH controller using the following sequence:

```
/* Enable single bit ECC error reporting in flash controller */
// Enable UTest mode
C55FMC.UT0.R = 0xF9F99999;
// Enable single bit error correction
C55FMC.UT0.B.SBCE = 1;
```

```
// Finish the UTest mode by writing UT0[UTE] with 0.
C55FMC.UT0.B.UTE = 0;
```

9.3.1 Option 1 - flash over programming

The approach described in Option 1 - flash over programming can be used. The chosen pattern for a single-bit injection may be as follows:

1. Write the original data A = 0xFFFFFFFF00000000 (syndrome X) to a flash memory location.

2. Over program data A to data B = 0xFFFFFFFF00000001 (syndrome Y) to the same flash memory location.

9.3.2 Option 2 - reading of UTEST area

The MPC57xx device family offers a set of test patterns pre-programmed in the factory (see Table 10). For a single-bit error, read address 0x00400040.

9.4 Intentional generation of Flash EDC after ECC error

The Flash EDC after ECC error reporting into the MEMU/ERM module is disabled by default (C55FMC_MCR[EEE] is not set on the EDC after ECC error event). It is needed to enable it by configuring the PFLASH controller as follows:

```
/* Enable single bit EDC after ECC error reporting in flash controller */
// Enable UTest mode
C55FMC.UT0.R = 0xF9F99999;
// Enable EDC after ECC Error Detection
C55FMC.UT0.B.CPE = 1;
// Finish the UTest mode by writing UT0[UTE] with 0.
C55FMC.UT0.B.UTE = 0;
```

The EDC after ECC error can only be injected by reading the test patterns pre-programmed in the factory (see Table 10). For the EDC after ECC error injection, read address 0x00400080.

10 Example code

10.1 MPC5744P 1b+2b RAM ECC error injection

The purpose of the example is to show how to generate multi-bit or single-bit ECC errors in the internal RAM (choose it in the option at the end of the main function).

The ECC fault is generated using core register E2EECSR. If the error injection is enabled (E2EECSR0[INVC]=1) and a certain mask is set (E2EECSR0[CHKINVT]), the subsequent write to the SRAM creates an error in the SRAM array.

When corrupted data is read, the IVOR1 exception handler is called in case of a multi-bit ECC error (IVOR1 exception occurs) and the FCCU_Alarm_Interrupt handler is called in case of a single-bit ECC error (an FCCU interrupt occurs). Both functions call the MEMU handler.

The example displays notes in the terminal window (connector J19 on MPC57xx_Motherboard, 19200-8-no parity-1 stop bit-no flow control on eSCI_A).

No other external connection is required.

10.2 MPC5744P 1b+2b PERRAM ECC error injection

The purpose of the example is to show how to simulate multi-bit or single-bit ECC errors in the internal DMA TCD RAM (choose it in the option at the end of the main function).

The Error Injection Module (EIM) is used to simulate multi-bit or single-bit ECC errors in the DMA TCD RAM (peripheral RAM).

When corrupted data is accessed, the IVOR1 exception handler is called in case of a multi-bit ECC error (IVOR1 exception occurs) and the FCCU_Alarm_Interrupt handler is called in case of a single-bit ECC error (an FCCU interrupt occurs). Both functions call the MEMU handler.

The example displays notes in the terminal window (connector J19 on MPC57xx_Motherboard, 19200-8-no parity-1 stop bit-no flow control on eSCI_A).

No other external connection is required.

10.3 MPC5744P 1b+2b Flash ECC error by UTEST read

The purpose of the example is to show how to generate multi-bit or single-bit ECC errors in the internal flash (choose it in the option at the end of the main function).

The ECC error is injected by reading pre-defined patterns in the UTEST area at addresses 0x00400040 and 0x00400060.

When corrupted data is accessed, the IVOR1 exception handler is called in case of a multi-bit ECC error (IVOR1 exception occurs) and the FCCU_Alarm_Interrupt handler is called in case of a single-bit ECC error (an FCCU interrupt occurs). Both functions call the MEMU handler.

The example displays notes in the terminal window (connector J19 on MPC57xx_Motherboard, 19200-8-no parity-1 stop bit-no flow control on eSCI_A).

No other external connection is required.

10.4 MPC5744P EDC after ECC error by UTEST read

The purpose of the example is to show how to generate the EDC after ECC error in the internal flash. An error response in achieved by reading pre-defined patterns in the UTEST area at address 0x00400080, which generates IVOR1 exception and FCCU interrupt (FCCU_Alarm_Interrupt).

The example does not show any handling, because it is application-specific.

The example displays notes in the terminal window (connector J19 on MPC57xx_Motherboard, 19200-8-no parity-1 stop bit-no flow control on eSCI_A).

No other external connection is required.

11 Revision history

Table 11. Revision history

Revision number	Date	Substantive changes
0	3 May 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 3 May 2021

Document identifier: AN13179

arm