1. Introduction

MPC57XX family of 32-bit microcontrollers is widely used in integrated automotive application controllers. It belongs to an expanding range of automotive focused products designed for flexibility to support a variety of applications. The advanced and cost-efficient host processor core of the MPC57XX (includes MPC5746R, MPC5775B/E, MPC5777C) automotive controller family complies with the Power Architecture® embedded category. All devices in this family are built around a safety concept based on the delayed lockstep, targeting an ISO26262 ASIL-D (Design) integrity level.

To meet the higher level of security needs, the safety requirements from the chip safety manual were complemented and were integrated in the safety library. For more detailed information about MPC57XX Safety Library please refer to the safety manual.

This application note introduces some functions of the MPC57XX Safety Library. It focuses on introducing the theory and implementation of flash ECC error report path test.
2. ECC and Flash memory characters

2.1. MPC57XX flash memory

The primary function of the embedded flash memory is to serve as an electrically programmable and erasable non-volatile memory (NVM). It can be used for instruction and data storage. The following figure shows the top-level diagram and functional organization of the flash memory unit.

The embedded flash memory consists of address spaces in four groupings.

- Low address space
- Mid address space
- High address space
- 256 KB address space

The main address space is divided into partitions. Each address space mentioned above consists of a partition pair. Partitions are used to determine locations for valid read-while-write (RWW) operations. While the embedded flash memory is performing a write (program or erase) operation to a given partition, it can simultaneously perform a read operation for any other partition. For program operations, only the address specified by an interlock write determines the partition being written (block locking and
block select registers do not determine the RWW partitions being written). For erase operations, only blocks that are selected and unlocked determine the RWW partitions being written.

The main address space is also divided into blocks to implement independent erase and program protection. The UTEST NVM block also exists as a block and has an independent program protection. The UTEST NVM block is included to support systems that require nonvolatile memory (NVM) for safety or to store system initialization information. Safety library shall use one block flash memory to execute the FLASH ECC error report path test. This block is also used to store safety information of other safety library test, for example, the MISRs of the Flash array integration test. This flash block shall be declared in the linker file and could not be used to store another data. And it also could not be the same partition with a flash of programming safety core binary.

The embedded flash memory is addressable by word (32 bits) or double word (64 bits) for program operation and page (256 bits) for a read operation. Multiple-words or double-word writes may be done in the flash memory to fill up the program page buffer (256 bits), enabling page programming (256 bits, requiring 4 double-word writes) and quad-page programming (1024 bits, requiring 16 double-word writes). Flash memory reads always return 256 bits, although read page buffering may be performed by the Bus Interface Unit (BIU).

Flash memory on-chip consists of a flash memory controller and a flash memory array module. The flash controller provides flash configuration and control functions and manages the interface between the flash memory array and the device crossbar switch. The following figure shows the flash memory architecture. The remaining sections give the functional details of the flash controller and flash array, followed by the memory maps.

![Flash memory block diagram](image)

**Figure 2. C55fmc memory block diagram**

### 2.2. ECC introduction

To support market demand related to improved functional and transient fault detection capabilities, this family of automotive microcontrollers includes end-to-end ECC (e2ECC) support. This e2ECC is
structurally different from traditional “ECC at memory” functionality because it provides robust error detection capabilities from one endpoint of an information transfer to another endpoint, with temporary information stored in one or more intermediate components.

Table 1. ECC comparison of data write-then-read sequence

<table>
<thead>
<tr>
<th>Traditional ECC</th>
<th>End-to-End ECC (e2eECC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus master initiates data write</td>
<td>Bus master initiates data write and generates ECC checkbits based on 29-bit address and 64-bit data fields for the computational shell, or based on 30-bit address and 32-bit data field for the peripheral shell</td>
</tr>
<tr>
<td>Data write transfer routed from bus master to appropriate bus slave</td>
<td>Data write transfer (including checkbits) routed from bus master to appropriate bus slave</td>
</tr>
<tr>
<td>For a bus memory slave, generate the ECC checkbits based on the data value and store data + checkbits into the memory</td>
<td>For a bus memory slave, store data and checkbits into the memory</td>
</tr>
<tr>
<td>Bus master initiates data read of previously written memory location</td>
<td>Bus master initiates data read of previously written memory location</td>
</tr>
<tr>
<td>Data read transfer routed from bus master to appropriate bus slave</td>
<td>Data read transfer routed from bus master to appropriate bus slave</td>
</tr>
<tr>
<td>For a bus memory slave, the memory array is accessed, the controller performs the ECC checkbit decode and syndrome generation, performs any needed single-bit correction and drives the read data onto the system bus interconnect</td>
<td>For a bus memory slave, the memory array is accessed, and the controller passes the read data and associated checkbits onto the system bus interconnection</td>
</tr>
<tr>
<td>The bus master captures the read data and continues</td>
<td>The bus master captures the read data and associated checkbits, performs the ECC checkbit decode and syndrome generation, performs any needed single-bit correction and continues</td>
</tr>
</tbody>
</table>

Memory protected by ECC/EDC traditionally generates and checks additional error parity information local to the memory unit to detect and/or correct errors that have occurred on data stored in the memory. On the other hand, e2eECC generates error protection codes at the source of data generation. It sends the encoded data and error protection codes to intermediate storage when a memory write is initiated by a bus master and performs a data integrity check using the previously-stored error protection codes at a data memory when a read of stored information is requested by a bus master. The intermediate storage may transform the generated error protection codes into another format for storage and then regenerate the codes for provision when a request is made to read the stored information or it may simply store the original protection codes unaltered, depending on the particular unit.

Additionally, the error protection codes are generated based on more than just the data associated with a storage location to protect additional information associated with access. In particular, address information corresponding to the access location of the stored information is combined with the stored data at the data source to generate error protection codes that cover certain types of addressing errors in the system interconnector or in the memory unit. The error protection codes may be checked at the memory unit on a store to ensure that no address information or data has been corrupted while the request has transitioned through the device from the bus master source. The error protection codes may also be checked to ensure that address decoding within the storage memory was performed properly (although not all address decoding errors can be detected this way) or it may simply store the received data and protection codes at the address it receives.

On a read request from a bus master, the memory unit retrieves the data information and error protection codes corresponding to the received address from the storage location or locations and supplies the data...
along with the error protection codes to the requesting device. The requesting device uses a locally stored address value corresponding to the read request to check the returning data and error protection codes to ensure that no errors have occurred in either addressing the memory or in the retrieved data. This ensures that the address sent for the request was not corrupted, the addressed location was actually accessed (to the extent it is possible to ensure) and the stored data was error-free, to the extent the ECC coding scheme can detect. As a result, the fault coverage provided by the end-to-end check is considerably more robust than the previous implementation of locally generated and checked/corrected error protection at each memory unit.

- Data check bits generation - external interfaces, 64-bit data ECC granularity
- Internal data check bits generation - 32-bit data ECC granularity
- The address portion of check bits generation, 64- or 32-bit data ECC granularity
- CACHE and IMEM data check bits generation - 64-bit ECC granularity
- D-CACHE and DMEM internal data check bits generation - 32-bit data ECC granularity

Figure 3. MPC5777C e2eECC overview

Single bit errors are automatically corrected and will be reported to the ERM/MEMU module. Multiple bit ECC errors will be detected and different platforms will cause different behavior. For z0, z1, z3, and z6 versions of e200 cores.
3. ECC error injection implementation

3.1. ECC error injection

Flash ECC error injection test includes two parts, single-bit error injection test and multi-bit error injection test. The safety library uses one flash block (the last one of the large block in this demo) to store some safety information for the flash ECC test. This block is divided into three parts and allocation is shown in Figure 4 and Figure 5. The first part is used to store important information (e.g., Flash ECC injection test header information). The second part and the third part are used for the flash ECC test. The second part is four-byte alignment, and the third part is eight-byte (8 bytes shall generate ECC) alignment. The second part is used to record the usage of this flash block and the third part is used for
ECC error injection. After completing an ECC error injection test, the second part shall be updated. The system will erase all this block when any one of the three blocks is full.

Figure 4. The flash block allocation

Figure 5. The ECC info store mechanism

3.1.1. **Inject ECC 1-bit error**

Flash single-bit ECC error needs to be programmed twice. Follow the steps for programming it:
1) Enable the single-bit ECC error report in the flash controller.

```c
static void FS_FLASH_AccessEnableCEReport(const uint8_t flashInst)
{
    C55FMC_Type *pBase = s_pFlashRegList[flashInst];
    /* Enable single bit ECC error reporting in flash controller */
    pBase->UT0 = 0xF9F99999;    /* Enable UTE */
    /*Enable single bit error correction*/
    pBase->UT0 = C55FMC_UT0_SBCE(1);
    pBase->UT0 = C55FMC_UT0_UTE(0); /* Disable UTE */
}
```

2) Program to the same valid flash address twice to generate a single-bit ECC error. The first time program value are 0xFFFFFFFF and 0x00000000; and the second time program value are 0xFFFFFFFF and 0x00000001.

```c
status = FS_FLASH_ProgramBytes(0xFFFFFFFF, 0x00000000);  
if (FS_STATUS_SUCCESS == status)
{
    status = FS_FLASH_ProgramBytes(0xFFFFFFFF, 0x00000001);
}
```

### 3.1.2. Inject ECC 2-bits error

Flash 2-bits ECC error also needs to be programed twice. Program the same valid flash address twice to generate a 2-bits ECC error. Follow the below steps:

1) The first time program value: 0x00450000 and 0x00000000;
2) The second time program value: 0x00580000 and 0x00000000.

```c
status = FS_FLASH_ProgramBytes(0x00450000, 0x00000000);  
if (FS_STATUS_SUCCESS == status)
{
    status = FS_FLASH_ProgramBytes(0x00580000, 0x00000000);
}
```

**NOTE**

There are some special data that might generate the same ECC, so you need to avoid these cases. The data with the same ECC check bits is shown in Table 4.
Table 4. Data with the same ECC

<table>
<thead>
<tr>
<th>Doubleword</th>
<th>Checkbits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF_FFFF_FFFF_FFFF</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_FFFF_FFFF_0000</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_FFFF_0000_FFFF</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_0000_FFFF_FFFF</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0000_FFFF_FFFF_FFFF</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_FFFF_0000_0000</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_0000_FFFF_0000</td>
<td>0xFF</td>
</tr>
<tr>
<td>0xFFFF_0000_0000_FFFF</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0000_FFFF_0000_0000</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0000_0000_0000_0000</td>
<td>0xFF</td>
</tr>
</tbody>
</table>

3.2. ECC error detection

When a flash ECC error occurred, the error is reported to MEMU(MPC5746R)/ERM(MPC5775E, MPC5775B, MPC5777C) and FCCU module. MEMU module can record flash error type(1-bit / 2-bits) and error address. ERM module just record flash error type(1-bit / 2-bits).

<table>
<thead>
<tr>
<th>NCF Index</th>
<th>Default Reaction Configuration at POR or Destructive Reset</th>
<th>Source</th>
<th>Error Reaction Path Check</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>No Reaction</td>
<td>MEMU</td>
<td>Reading preprogrammed correctable ECC error from UTEST flash memory + Via FCCU Fake bit to set the fault at error path + LBIST</td>
<td>Flash Correctable Error detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NCF correlated to C55FMC_MCR[SBC] bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NOTE: From the MEMU perspective this is indicated by MEMU_FLASH_CERR_STSn. For details, see the description of the C55FMC_MCR and MEMU_FLASH_CERR_STS registers.</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>No Reaction</td>
<td>MEMU</td>
<td>Reading preprogrammed uncorrectable ECC error from UTEST flash memory + Via FCCU Fake bit to set the fault at error path + LBIST</td>
<td>Flash Un-Correctable Error detected</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NCF correlated to C55FMC_MCR[EEER] bit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NOTE: From the MEMU perspective this is indicated by MEMU_FLASH_UNCERR_STSn. For details, see the description of the C55FMC_MCR and MEMU_FLASH_UNCERR_STS registers.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 6. MPC5746R FCCU channel for flash
MPC5775E/ MPC5775B/MPC5777C FCCU channel for flash ECC error.

![Table](image)

Figure 7. MPC577x FCCU channel for flash

MPC5746R MEMU has 20 entries for flash 1-bit error and 1 entry for flash 2-bits error. Each entry in the reporting table corresponds to a unique error event.

<table>
<thead>
<tr>
<th>Error source</th>
<th>Number of entries in correctable error reporting table</th>
<th>Number of entries in un-correctable error reporting table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flash memory</td>
<td>20</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 8. MPC5746R MEMU entry for flash

MPC5775E/ MPC5775B/MPC5777C ERM channel for flash ECC error

![Table](image)

Figure 9. MPC577x ERM channel for flash

### 3.3. Software Implementation

For software implementation follow the below steps:

1. Fetching the flash ECC injection address;
   
   ```c
   FS_FLASH_CalcEccErrReportAddr(&singleBitAddr, &multiBitAddr);
   ```

2. Disable data cache;

   ```c
   FS_CACHE_Disable(FS_CACHE_TYPE_DCACHE);
   ```

3. 2-bit flash ECC error will lead to IVOR1, need to install exception call back function to jump out IVOR1.

   ```c
   FS_EXCEPTION_RegisterCallback(...);
   ```

### 3.3.1 ECC 1-bit software implementation

The software procedure (initialization, injection, and check error flag) is completed by the steps shown in the following figure.
3.3.2 ECC 2-bits software implementation

The software procedure (initialization, injection, and check error flag) is being done by the steps shown in the following figure.
4. Demo code and test results

The purpose of the demo is to show how to inject 1-bit and 2-bit ECC error in internal FLASH (you can choose the 1-bit or 2-bit operation depend on the special application) and how to check flash ECC error with safety library base on MPC57XX MCUs. The demo is only possible to run in the internal FLASH target and the test block is invalid for application. Ensure the UART is working normally because all the test information will be printed through it.

4.1. Test logic and code

The whole sequence (test initialization, injection, handling and print result) shown in the following steps:

1. Definition a test result variable for test.
   
   ```c
   fs_flash_ecc_report_path_test_result_t testObj;
   ```

2. Definition and initialize a test input variable for test.
   
   ```c
   fs_flash_ecc_report_path_test_param_t testCfg = {
     .singleBitSelect = 1;
     .multiBitSelect = 1;
   };
   ```

3. Configure the flash ECC test address in the linker file.
   
   ```c
   FS_FLASH_ECC_ERR_PATH_TEST_ADDR = 0xBC0000;
   ```
4. Set the PIT timer 0 as delay timer.
   
   \[ \text{FS}_{\text{TIMER}}._{	ext{DelayInit}}(0); \]

5. Initialize store safety information.
   
   \[ \text{FS}_{\text{STORE}}._{\text{SafeInfoStoreInit}}(); \]

6. Initialize the flash ECC error report path test configuration.
   
   \[ \text{FS}_{\text{FLASH}}._{\text{GetEccReportPathTestConfig}}(&\text{testCfg}); \]

7. Execute flash ECC error report path test.
   
   \[ \text{FS}_{\text{FLASH}}._{\text{EccReportPathTest}}(&\text{testObj}); \]

8. Check and print out test result.
   
   \[
   \begin{align*}
   &\text{if}(!\text{testObj.singleBitPassed}) \\
   &\{ \\
   &\text{FS}_{\text{PRINT}}._{\text{ERR}}("Failed to test Flash ECC single bit Err path.\n"); \\
   &\}
   &\text{if}(!\text{testObj.multiBitPassed}) \\
   &\{ \\
   &\text{FS}_{\text{PRINT}}._{\text{ERR}}("Failed to test Flash ECC multi bit Err path.\n"); \\
   &\}
   \end{align*}
   \]

4.2. Test results

UART is used to print the result of the Flash ECC error report path test.

![Figure 7. Flash ECC error report path test result](image)

5. References

- MPC57XX PRM
- AN5288
- AN5200
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, C 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C Ware, the Energy Efficient Solutions logo, Kinetics, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airstaf, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, AMBA, Arm Powered, Artisan, Cortex, Jazelle, Keil, SecurCore, Thumb, TrustZone, and uVision are registered trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. Arm7, Arm9, Arm11, big.LITTLE, CoreLink, CoreSight, DesignStart, Mali, Mbed, NEON, POP, Sensinode, Socrates, ULINK and Versatile are trademarks of Arm Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2020 NXP B.V.