ECC 功能使用指南

概述

本手册适用范围：

适用范围	
SPC1125 系列	SPC1125，SPC1128
SPC1168 系列	SPC1155， SPC1156， SPC1158， SPC1168，SPD1148， SPD1178， SPD1188， SPD1163，SPM1173
SPC2168 系列	SPC2168，SPC2165，SPC2166，SPC1198
SPC1169 系列	SPC1169，SPD1179，SPD1176
SPC2188 系列	SPC1185，SPC2188

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
C/0	2024-04-24	何序清	Outdated	首次发布。
C/1	2024-06-27	苏杭	Outdated	1. 更新章节 1，增加章节 2。
C/2	2024-08-06	苏杭	Released	1. 更新章节 1，增加章节 3。

术语或缩写

术语或缩写	描述
MCU	Microcontroller Unit，微控制器单元

1 特性

ECC（Error Correction Code）是一种广泛应用于数据存储和传输过程中的错误检测与修正机制。数据写入过程在原始数据上附加一个额外的纠错码，从而能够在读取过程中纠正单比特错误和检测多比特错误。

2 功能描述

数据写入过程在原始数据上附加一个额外的纠错码，从而能够在读取过程中纠正单比特错误和检测多比特错误。ECC 纠错码虽然不可读，但设计了中断来检测 ECC 错误，如表 2-1 所示。

表 2-1：ECC 检测

芯片	触发复位	中断	错误地址记录
SPC1169 系列	可配	ECC 发生多 Bit 错误时会向 MCU 发出 NMI 中断，而发生单 Bit 错误时会向 MCU 发出 MEM 中断。	发生位错误的地址将记录在特殊寄存器中，以供诊断使用。
SPC2188 系列	可配	向 MCU 发出 NMI 中断。	发生位错误的地址将记录在特殊寄存器中，以供诊断使用。
SPC1168 系列	可配	向 MCU 发出 MEM 中断。	无对应功能。
SPC1128 系列	可配	向 MCU 发出 MEM 中断。	发生位错误的地址将记录在特殊寄存器中，以供诊断使用。
SPC2168 系列	可配	向 MCU 发出 MEM 中断。	无对应功能。

凡是具有 ECC 功能的模块，默认 ECC 都是打开的，但具体是否可关，如表 2-2 所示。

表 2-2：ECC 开关状态

芯片	区域	ECC 开关是否可配	ECC 状态
SPC1169 系列	Flash ROM RAM	不可配	永远开
SPC2188 系列	ROM RAM	不可配	永远开
	Flash	可通过 0x11001000 扇区的 TRIM 数据配置关闭	默认开
SPC1168 系列，SPC1128 系列	Flash ROM	可通过 MEMECCEN 配置关闭	默认开
SPC2168 系列	Flash ROM RAM	可通过 MEMECCEN 配置关闭	默认开

注意： SPC1168 系列及 SPC1128 系列的 RAM 不支持 ECC 功能，但支持奇偶校验。

存在特殊 Flash 区域，不具有 ECC 功能，如表 2-3 所示。

表 2-3：Flash 中不含 ECC 功能的区域

芯片	Flash 中不含 ECC 功能的区域
SPC1169 系列	Flash 整个区域包含 ECC 功能
SPC2188 系列	OTP 区域不含 ECC 功能
SPC1168 系列，SPC1128 系列	Flash 整个区域包含 ECC 功能
SPC2168 系列	Flash 整个区域包含 ECC 功能

部分芯片空间大小受 ECC 开关影响，如表 2-4 所示。

表 2-4：ECC 对芯片区域大小的影响

芯片	默认 ECC 开启时	ECC 关闭后
SPC1185，SPC2188 Main Flash 大小	2K	4K
SPC2188 Main Flash 大小	1M	2M
SPC1185 Main Flash 大小	256KB	512KB

Flash 擦除后（此时 Data 以及 ECC 全 1），直接读取，理论会报 ECC 错误（因为 ECC 校验不通过），但为了方便使用，会屏蔽 Flash 中 Data 以及 ECC 全 1 时的 ECC 错误，如表 2-5 所示。

表 2-5：是否屏蔽 Data 以及 ECC 全 1 时 Flash ECC 检测

芯片	是否屏蔽 Data 以及 ECC 全 1 时 Flash ECC 检测
SPC1169 系列	是
SPC2188 系列	是
SPC1168 系列，SPC1128 系列	是
SPC2168 系列	是

存在特殊 Flash 读函数，会在调用时临时屏蔽 ECC 错误检测，如表 2-6 所示。

表 2-6：屏蔽 ECC 检测的特殊读函数

芯片	屏蔽 ECC 检测的特殊读函数
SPC1169 系列	FLASHC_VerifyRead FLASHC_VerifyErase
SPC2188 系列	无
SPC1168 系列	无
SPC2168 系列	无
SPC1128 系列	FLASH_VerifyProgramRead FLASH_VerifyEraseRead FLASH_VerifyErase

3 功能实例

3.1 SPC2188 系列

3.1.1 Flash ECC 功能验证

3.1.1.1 功能需求

验证 Flash ECC 功能有效。

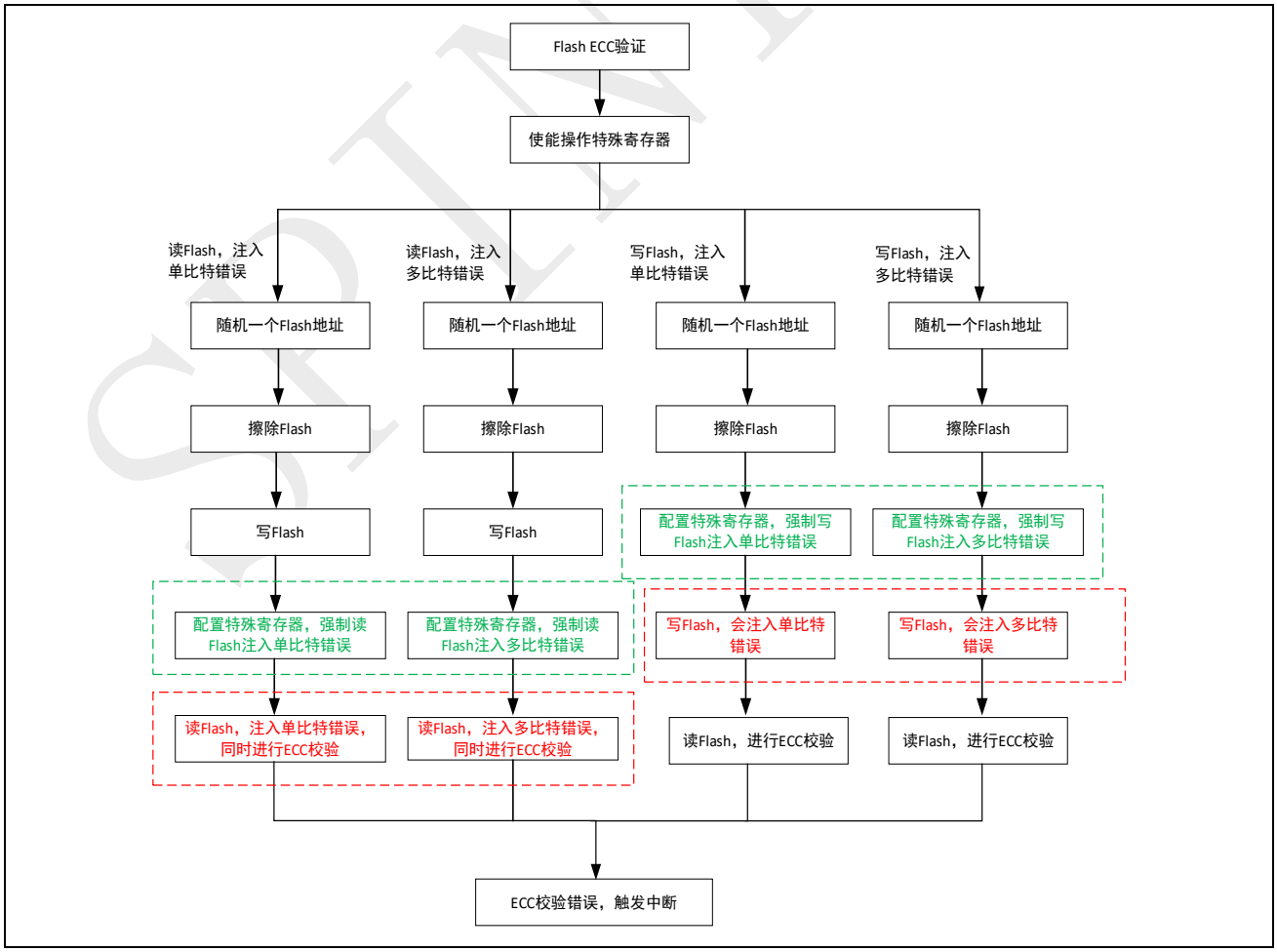
3.1.1.2 功能实现

本示例演示通过强制给 Flash 注入单比特错误与多比特错误，产生 ECC 校验错误中断来验证 Flash 具有 ECC 功能。由于需要验证 Flash 的 ECC 功能，所以验证的测试程序不能存放在 Flash 中，需要放在 RAM 中运行，对 Flash 的 ECC 工程验证的程序执行流程如图 3-1 所示。

在读 Flash 注入错误的流程中的绿色框配置，是使能读 Flash 时将会把 Flash 的 1 个比特或者 2 个比特的数据读错，从而在红色框进行读 Flash 时，会产生 ECC 校验错误。

在写 Flash 注入错误的流程中的绿色框配置，是使能写 Flash 时将会把 Flash 的 1 个比特或者 2 个比特的数据写错，从而在红色框进行写 Flash 时，将写错 1 个比特或者 2 个比特，最后进行读 Flash 时，会产生 ECC 校验错误。

图 3-1：Flash 的 ECC 验证流程



示例配置程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 Flash；
- 使能产生单比特错误和多比特错误时产生不可屏蔽中断；
- 使能操作特殊寄存器；
- 随机出一个特殊的 Flash 地址进行访问；
- 擦除随机出来的 Flash 地址所在扇区；
- 随机出一个比特或者两个比特进行单比特错误或者多比特错误；
- 如果是写 Flash 注入错误，则操作特殊寄存器进行使能写 Flash 注入错误；
- 写 flash；
- 如果是读 Flash 注入错误，则操作特殊寄存器进行使能读 Flash 注入错误；
- 读 Flash。

Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "SPC2188.h"


/* Special macro definitions */
#define READ_FLASH_ERROR   (1U << 1 )      /* Error when read FLASH */
#define WRITE_FLASH_ERROR  (1U << 17)      /* Error when write FLASH */

uint32_t    gu32IntCnt = 0;
uint32_t    gu32IntEvt;
uint32_t    gu32Addr;

volatile uint32_t u32Data;

void FLASH_AccessErrorIntCheck(uint32_t u32Base, uint32_t u32Size, uint8_t
u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;
    uint8_t     u8Buf;

    gu32IntCnt = 0U;

    /* Get random addr */
    switch (rand() & 7U)
    {
        /* 1/8 propability */
        case 0U:
```

Example Code

```
                gu32Addr = u32Base;
                break;
        /* 1/8 propability */
        case 1U:
                gu32Addr = u32Base + u32Size - 1U;
                break;
        /* 6/8 propability */
        default:
                gu32Addr = ((rand() % u32Size) + u32Base);
                break;
    }

    /* Erase flash Sector */
    FLASH_EraseSector(QSPI1, gu32Addr);

    u32Bit0 = rand() % 13U;
    if (u8ErrorBitNum == 2U)
    {
        do
        {
            u32Bit1 = rand() % 13U;
        } while (u32Bit0 == u32Bit1);
    }
    else
    {
        u32Bit1 = u32Bit0;
    }
    u8Buf = rand();
    if (u32Access == WRITE_FLASH_ERROR)
    {
        /* Force bit error pattern */
        WRITE_REG(*((volatile uint32_t*)(0x40000F30)), (1U << u32Bit0) | (1U <<
u32Bit1));
        /* Force memory access error */
        SET_BITS(*((volatile uint32_t*)(0x40000F3C)), u32Access);
    }

    /* Write Flash Data */
    FLASH_Program(QSPI1, &u8Buf, gu32Addr, 1);

    if (u32Access == READ_FLASH_ERROR)
    {
        u32Bit0 = rand() % 13U;
        if (u8ErrorBitNum == 2U)
        {
            do
            {
                u32Bit1 = rand() % 13U;
            } while (u32Bit0 == u32Bit1);
```

Example Code

```
        }
        else
        {
            u32Bit1 = u32Bit0;
        }

        /* Force bit error pattern */
        WRITE_REG(*((volatile uint32_t*)(0x40000F30)), (1U << u32Bit0) | (1U <<
u32Bit1));
        /* Force memory access error */
        SET_BITS(*((volatile uint32_t*)(0x40000F3C)), READ_FLASH_ERROR);

        /* Read flash Data */
        u32Data = *((__IO uint8_t *)gu32Addr);
        while (gu32IntCnt == 0);
    }
    else
    {
        /* Read flash Data */
        u32Data = *((__IO uint8_t *)gu32Addr);
        while (gu32IntCnt == 0);
    }
}

int main( void )
{

    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);


    /* When using SWD for RAM simulation, boot does not go through the
initialization flash process */
    if((QSPI_RecallFlashID(QSPI1) >> 16) == 0xC8U)
    {
        FLASH_Init(QSPI1, SysInfo.u32SYSCLK0, 80000000);
        FLASH_SetTiming(QSPI1, 240000000U, 80000000);
    }
    else
    {
        FLASH_Init(QSPI1, SysInfo.u32SYSCLK0, 50000000);
```

Example Code

```
        FLASH_SetTiming(QSPI1, 240000000U, 50000000);
    }


    printf("Flash ECC test\n");

    SYSTEM_DisableResetEvent(RESET_EVENT_ALL);
    SYSTEM_DisableNonMaskableInt(NMI_EVENT_ALL);
    SYSTEM_EnableNonMaskableInt (NMI_EVENT_FLASH_SINGLE_BIT_ERROR |
NMI_EVENT_FLASH_MULTI_BIT_ERROR);

    /* Enable write special registers */
    WRITE_REG(*((volatile uint32_t*)(0x40000F54)), 0x1ACCE551U);

    /* Read flash ECC single bit error */
    gu32IntEvt = NMI_EVENT_FLASH_SINGLE_BIT_ERROR;
    FLASH_AccessErrorIntCheck(0x10010000, 0x1000, 1, READ_FLASH_ERROR);

    /* Read flash ECC multi bit error */
    gu32IntEvt = NMI_EVENT_FLASH_MULTI_BIT_ERROR;
    FLASH_AccessErrorIntCheck(0x10010000, 0x1000, 2, READ_FLASH_ERROR);

    /* write flash ECC single bit error */
    gu32IntEvt = NMI_EVENT_FLASH_SINGLE_BIT_ERROR;
    FLASH_AccessErrorIntCheck(0x10010000, 0x1000, 1, WRITE_FLASH_ERROR);

    /* Write flash ECC multi bit error */
    gu32IntEvt = NMI_EVENT_FLASH_MULTI_BIT_ERROR;
    FLASH_AccessErrorIntCheck(0x10010000, 0x1000, 2, WRITE_FLASH_ERROR);

    while(1) {}

}

void NMI_Handler(void)
{
    uint32_t u32ErrorAddr = 0;

    switch(gu32IntEvt)
    {
        case (NMI_EVENT_FLASH_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_FLASH);
            printf("NMI_EVENT_FLASH_SINGLE_BIT_ERROR\n");
            break;
        case (NMI_EVENT_FLASH_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_FLASH);
            printf("NMI_EVENT_FLASH_MULTI_BIT_ERROR\n");
            break;
    }
```

Example Code

```
    if (u32ErrorAddr != gu32Addr)
    {
        printf("ECC error should be detected at 0x%08x, not 0x%08x\n", gu32Addr,
u32ErrorAddr);
    }
    else
    {
        printf("ECC detected success\n");
    }

    gu32IntCnt++;
    SYSTEM_ClearNonMaskableInt(NMI_EVENT_GLOBAL | gu32IntEvt);
}
```

3.1.2 ROM、RAM ECC 功能验证

3.1.2.1 功能需求

验证 ROM、RAM ECC 功能有效。

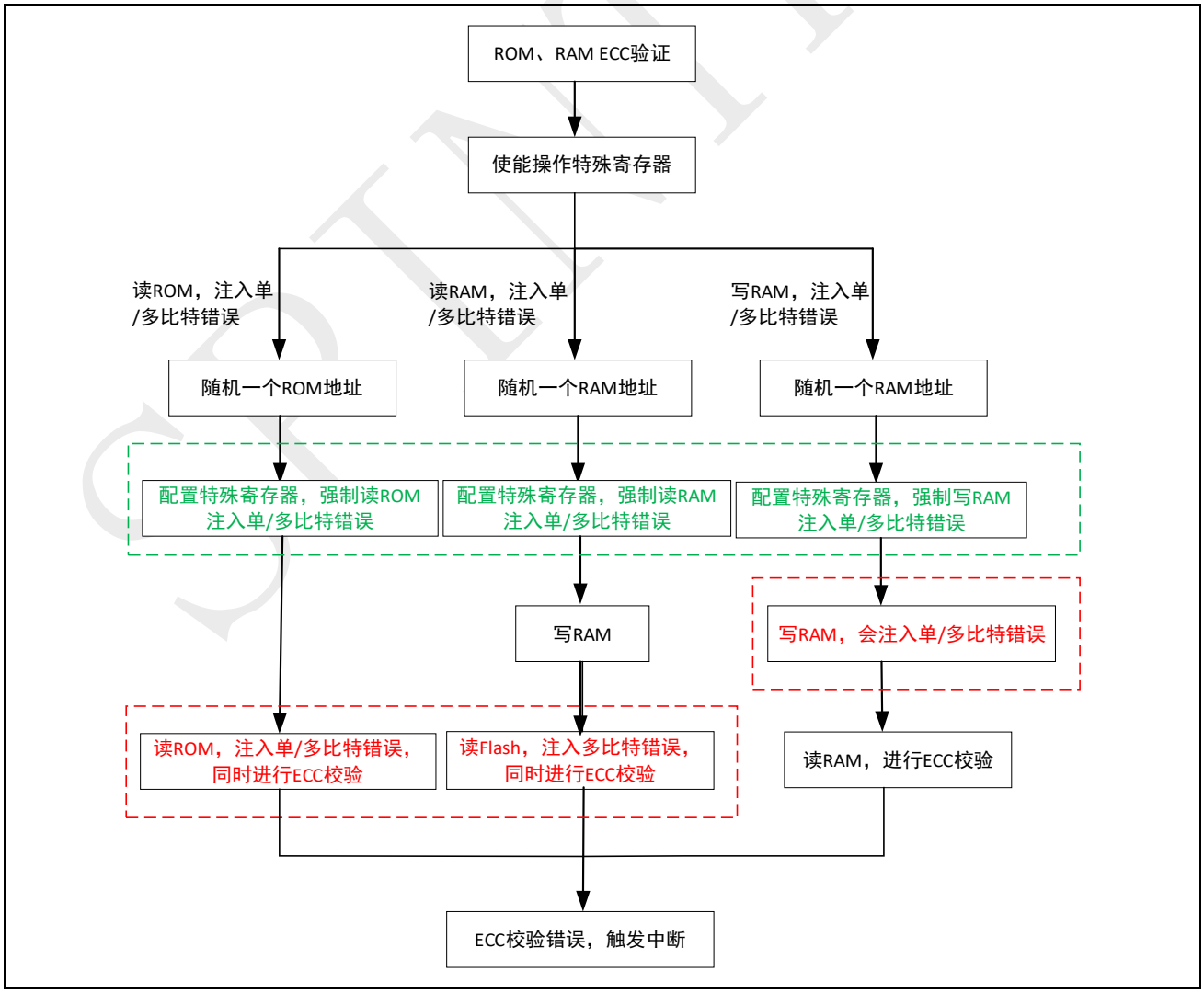
3.1.2.2 功能实现

本示例演示通过强制给 ROM、RAM 注入单比特错误与多比特错误，产生 ECC 校验错误中断来验证 ROM、RAM 具有 ECC 功能。由于需要验证 ROM、RAM 的 ECC 功能，所以验证的测试程序不能存放在 RAM 中，需要放在 Flash 中运行，对 ROM、RAM 的 ECC 工程验证的程序执行流程如图 3-2 所示。

在读 ROM 或者 RAM 注入错误的流程中的绿色框配置，是使能读 ROM 或者 RAM 时将会把 ROM 或者 RAM 的 1 个比特或者 2 个比特的数据读错，从而在红色框进行读 ROM 或者 RAM 时，会产生 ECC 校验错误。

在写 RAM 注入错误的流程中的绿色框配置，是使能写 RAM 时将会把 RAM 的 1 个比特或者 2 个比特的数据写错，从而在红色框进行写 RAM 时，将写错 1 个比特或者 2 个比特，最后进行读 RAM 时，会产生 ECC 校验错误。

图 3-2：ROM、RAM 的 ECC 验证流程



示例配置程序配置流程如下：

- 初始化系统时钟，UART 调试口；
- 使能产生单比特错误和多比特错误时产生不可屏蔽中断；
- 使能操作特殊寄存器；
- 随机出一个特殊的 ROM 与 RAM 地址进行访问；
- 随机出一个比特或者两个比特进行单比特错误或者多比特错误；
- 如果是写 RAM 注入错误，则操作特殊寄存器进行使能写 RAM 注入错误；
- 如果是读 ROM 或者 RAM 注入错误，则操作特殊寄存器进行使能读 ROM 或者 RAM 注入错误；
- 如果是 RAM 地址，则进行写 RAM 操作，因为 ROM 不能进行写操作，只能读操作验证 ECC；
- 读 RAM 或者 ROM 操作。

Example Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "SPC2188.h"

/* Special macro definitions */
#define READ_ROM_ERROR      (1U << 0 )   /* Error when read ROM */
#define READ_RAM0_ERROR     (1U << 2 )   /* Error when read RAM0 */
#define WRITE_RAM0_ERROR    (1U << 18)   /* Error when write RAM0 */

uint32_t    gu32IntCnt = 0;
uint32_t    gu32IntEvt;
uint32_t    gu32Addr;


void ROM_ReadErrorIntCheck(uint32_t u32Base, uint32_t u32Size, uint8_t
u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;

    gu32IntCnt = 0U;

    /* Get random addr */
    switch (rand() & 7U)
    {
        /* 1/8 propability */
        case 0U:
                gu32Addr = u32Base;
```

Example Code

```
                break;
        /* 1/8 propability */
        case 1U:
                gu32Addr = u32Base + u32Size - 1U;
                break;
        /* 6/8 propability */
        default:
                gu32Addr = ((rand() % u32Size) + u32Base);
                break;
    }

    if (u8ErrorBitNum == 1U)
    {
        u32Bit0 = rand() % 39U;
        /* Force bit error pattern */
        WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)), 1U
<< (u32Bit0 & 31U));
    }

    if (u8ErrorBitNum == 2U)
    {
        do
        {
            u32Bit0 = rand() % 39U;
            u32Bit1 = rand() % 39U;
        } while (u32Bit0 == u32Bit1);
        if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
        {
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)),
(1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
        }
        else
        {
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)),
1U << (u32Bit0 & 31U));
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit1 >> 5) * 4)),
1U << (u32Bit1 & 31U));
        }
    }

    /* Force memory access error */
    SET_BITS(*((volatile uint32_t*)(0x40000F3C)), u32Access);

    /* Read ROM data */
    SYSTEM_SetNVREG1(*((__IO uint32_t*)(uint32_t)gu32Addr));

    while (gu32IntCnt == 0);
}
```

Example Code

```
void RAM_AccessErrorIntCheck(uint32_t u32Base, uint32_t u32Size, uint8_t
u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    u32Byte;
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;

    gu32IntCnt = 0U;

    /* Get random addr */
    switch (rand() & 7U)
    {
        /* 1/8 propability */
        case 0U:
               gu32Addr = u32Base;
               break;
        /* 1/8 propability */
        case 1U:
               gu32Addr = u32Base + u32Size - 1U;
               break;
        /* 6/8 propability */
        default:
               gu32Addr = ((rand() % u32Size) + u32Base);
               break;
    }

    gu32Addr &= 0xfffffffcU;

    /* Save addr and data */
    SYSTEM_SetNVREG0(gu32Addr);
    SYSTEM_SetNVREG1(rand());

    if (u8ErrorBitNum == 1U)
    {
        u32Bit0 = rand() % 13U;
        u32Bit0 += (gu32Addr & 0x3) * 13U;
        u32Bit0 += (rand() % (1 << 2)) * 13U;
        /* Force bit error pattern */
        WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)), 1U
<< (u32Bit0 & 31U));
    }

    if (u8ErrorBitNum == 2U)
    {
        do
        {
            u32Byte = rand() % (1 << 2);
```

Example Code

```
            u32Bit0 = rand() % 13U;
            u32Bit1 = rand() % 13U;
            u32Bit0 += (gu32Addr & 0x3) * 13U;
            u32Bit1 += (gu32Addr & 0x3) * 13U;
            u32Bit0 += u32Byte * 13U;
            u32Bit1 += u32Byte * 13U;
        } while (u32Bit0 == u32Bit1);

        if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
        {
            /* Force bit error pattern */
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)),
(1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
        }
        else
        {
            /* Force bit error pattern */
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit0 >> 5) * 4)),
1U << (u32Bit0 & 31U));
            WRITE_REG(*((volatile uint32_t*)(0x40000F30 + (u32Bit1 >> 5) * 4)),
1U << (u32Bit1 & 31U));
        }
    }

    /* Force memory access error */
    SET_BITS(*((volatile uint32_t*)(0x40000F3C)), u32Access);

    /* write RAM and read data */
    *((__IO uint32_t*)(uint32_t)SYSTEM_GetNVREG0()) = SYSTEM_GetNVREG1();
    SYSTEM_SetNVREG0(*((__IO uint32_t*)(uint32_t)SYSTEM_GetNVREG0()));
}


int main( void )
{

    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("ROM RAM ECC test\n");
```

Example Code

```
    SYSTEM_DisableResetEvent(RESET_EVENT_ALL);
    SYSTEM_DisableNonMaskableInt(NMI_EVENT_ALL);
    SYSTEM_EnableNonMaskableInt (
                                NMI_EVENT_ROM_SINGLE_BIT_ERROR
                              | NMI_EVENT_ROM_MULTI_BIT_ERROR
                              | NMI_EVENT_RAM0_SINGLE_BIT_ERROR
                              | NMI_EVENT_RAM0_MULTI_BIT_ERROR
                              | NMI_EVENT_RAM1_SINGLE_BIT_ERROR
                              | NMI_EVENT_RAM1_MULTI_BIT_ERROR
                              | NMI_EVENT_RAM2_SINGLE_BIT_ERROR
                              | NMI_EVENT_RAM2_MULTI_BIT_ERROR
                              );

    /* Enable write special registers */
    WRITE_REG(*((volatile uint32_t*)(0x40000F54)), 0x1ACCE551U);


    /* Read ROM ECC single bit error */
    gu32IntEvt = NMI_EVENT_ROM_SINGLE_BIT_ERROR;
    ROM_ReadErrorIntCheck   (0x00001000U, 0x40, 1, READ_ROM_ERROR);

    /* Read ROM ECC multi bit error */
    gu32IntEvt = NMI_EVENT_ROM_MULTI_BIT_ERROR;
    ROM_ReadErrorIntCheck   (0x00001000U, 0x40, 2, READ_ROM_ERROR);

    /* Read RAM0 ECC single bit error */
    gu32IntEvt = NMI_EVENT_RAM0_SINGLE_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1ffb0000U, 0x10000, 1, READ_RAM0_ERROR);

    /* Read RAM0 ECC multi bit error */
    gu32IntEvt = NMI_EVENT_RAM0_MULTI_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1ffb0000U, 0x10000, 2, READ_RAM0_ERROR);

    /* Write RAM0 ECC single bit error */
    gu32IntEvt = NMI_EVENT_RAM0_SINGLE_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1ffb0000U, 0x10000, 1, WRITE_RAM0_ERROR);

    /* Write RAM0 ECC multi bit error */
    gu32IntEvt = NMI_EVENT_RAM0_MULTI_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1ffb0000U, 0x10000, 2, WRITE_RAM0_ERROR);

    while(1) {}

}

void NMI_Handler(void)
{
    uint32_t u32ErrorAddr = 0;
```

Example Code

```
    switch(gu32IntEvt)
    {
        case (NMI_EVENT_ROM_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_ROM);
            printf("NMI_EVENT_ROM_SINGLE_BIT_ERROR\n");
            break;
        case (NMI_EVENT_ROM_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_ROM);
            printf("NMI_EVENT_ROM_MULTI_BIT_ERROR\n");
            break;
        case (NMI_EVENT_RAM0_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM0);
            printf("NMI_EVENT_RAM0_SINGLE_BIT_ERROR\n");
            break;
        case (NMI_EVENT_RAM0_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM0);
            printf("NMI_EVENT_RAM0_MULTI_BIT_ERROR\n");
            break;
    }

    if (u32ErrorAddr != gu32Addr)
    {
        printf("ECC error should be detected at 0x%08x, not 0x%08x\n", gu32Addr,
u32ErrorAddr);
    }
    else
    {
        printf("ECC detected success\n");
    }

    gu32IntCnt++;

    SYSTEM_ClearNonMaskableInt(NMI_EVENT_GLOBAL | gu32IntEvt);
}
```

3.2 SPC1168/SPC1125 系列

3.2.1 Flash ECC 功能验证

3.2.1.1 功能需求

验证 Flash ECC 功能有效。

3.2.1.2 功能实现

本示例演示通过强制给 Flash 注入单比特错误与多比特错误，产生 ECC 校验错误中断来验证 Flash 具有 ECC 功能。由于需要验证 Flash 的 ECC 功能，所以验证的测试程序不能存放在 Flash 中，需要放在 RAM 中运行。

示例配置程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 Flash；
- 使能操作特殊寄存器；
- 清除单比特错误和多比特错误标志位；
- 使能产生单比特错误和多比特错误以及 Flash ECC 校验；
- 擦除 Flash 地址所在扇区；
- 如果是写 Flash 注入错误，则操作特殊寄存器进行使能写 Flash 注入错误；
- 如果是读 Flash 注入错误，则操作特殊寄存器进行使能读 Flash 注入错误；
- 写 flash；
- 读 Flash。

Example Code

```
uint32_t u32Data;

int main(void)
{

    /* Set Flash timing */
    FLASH_WALLOW();

    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(100000000UL);

    /* Delay init */
    Delay_Init();

    /* UART init */
```

Example Code

```
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);

    UART_Init(UART, 38400);

    printf("Just a Sample....\n");


    FLASH_WALLOW();
    /* ENGR */
    WRITE_REG(*((volatile uint32_t*)(0x40000F50)), 0x1ACCE551U);

    NVIC_EnableIRQ(MEM_IRQn);

    SYSTEM->MEMIC.bit.FLASH1ERR = 1;
    SYSTEM->MEMIC.bit.FLASH2ERR = 1;
    SYSTEM->MEMECCEN.bit.FLASHECC = 1;
    SYSTEM->MEMIE.bit.FLASH1ERR = 1;
    SYSTEM->MEMIE.bit.FLASH2ERR = 1;

    /* write flash, generate 1bit error */
    {
        FLASH_EraseSector(0x10000000);
        WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x1e << 12);
        FLASH_ProgramWord(0xa5a5, 0x10000000);
        u32Data = *((__IO uint32_t *)0x10000000);
    }

    /* write flash, generate 2bit error */
    {
        FLASH_EraseSector(0x10000000);
        WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x2e << 12);
        FLASH_ProgramWord(0xa5a5, 0x10000000);
        u32Data = *((__IO uint32_t *)0x10000000);
    }

    /* read flash, generate 1bit error */
    {
        FLASH_EraseSector(0x10000000);
        WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x1e << 6);
        FLASH_ProgramWord(0xa5a5, 0x10000000);
        u32Data = *((__IO uint32_t *)0x10000000);
    }

    /* read flash, generate 2bit error */
    {
        FLASH_EraseSector(0x10000000);
        WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x2e << 6);
```

Example Code

```
        FLASH_ProgramWord(0xa5a5, 0x10000000);
        u32Data = *((__IO uint32_t *)0x10000000);
    }

    while (1)
    {

    }
}

void MEM_IRQHandler(void)
{
    if (SYSTEM->MEMIF.bit.FLASH1ERR == 1)
    {
        printf("NMI_EVENT_FLASH_SINGLE_BIT_ERROR\n");
        SYSTEM->MEMIC.bit.FLASH1ERR = 1;
    }

    if (SYSTEM->MEMIF.bit.FLASH2ERR == 1)
    {
        printf("NMI_EVENT_FLASH_MULTI_BIT_ERROR\n");
        SYSTEM->MEMIC.bit.FLASH2ERR = 1;
    }

    SYSTEM->MEMIC.bit.MEMINT = 1;
}
```

[1] 示例代码适用于 SPC1168，其它系列产品的示例代码会根据实际需求进行补充。

3.3 SPC2168 系列

3.3.1 Flash ECC 功能验证

3.3.1.1 功能需求

验证 Flash ECC 功能有效

3.3.1.2 功能实现

本示例演示通过强制给 Flash 注入单比特错误与多比特错误，产生 ECC 校验错误中断来验证 Flash 具有 ECC 功能。由于需要验证 Flash 的 ECC 功能，所以验证的测试程序不能存放在 Flash 中，需要放在 RAM 中运行。

示例配置程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 Flash；
- 使能操作特殊寄存器；
- 清除单比特错误和多比特错误标志位；
- 使能产生单比特错误和多比特错误以及 Flash ECC 校验；
- 擦除随机出来的 Flash 地址所在扇区；
- 如果是写 Flash 注入错误，则操作特殊寄存器进行使能写 Flash 注入错误；
- 如果是读 Flash 注入错误，则操作特殊寄存器进行使能读 Flash 注入错误；
- 写 flash；
- 读 Flash。

Example Code

```
uint32_t u32Data;

int main(void)
{
  /* Set Flash timing */
  FLASH_WALLOW();
  FLASH_SetTiming(200000000);
  FLASH_WDIS();

  /* Clock init */
  CLOCK_InitWithRCO(200000000UL);

  /* Delay init */
  Delay_Init();

  /* UART init */
  GPIO_SetPinChannel(GPIO_44, GPIO44_UART_TXD);
  GPIO_SetPinChannel(GPIO_45, GPIO45_UART_RXD);
```

Example Code

```
UART_Init(UART, 38400);

printf("Just a Sample....\n");

  FLASH_WALLOW();
  /* ENGR */
  WRITE_REG(*((volatile uint32_t*)(0x40000F50)), 0x1ACCE551U);


  NVIC_EnableIRQ(MEM_IRQn);

  SYSTEM->MEMIC.bit.FLASH1ERR = 1;
  SYSTEM->MEMIC.bit.FLASH2ERR = 1;
  SYSTEM->MEMECCEN.bit.FLASHECC = 1;
  SYSTEM->MEMIE.bit.FLASH1ERR = 1;
  SYSTEM->MEMIE.bit.FLASH2ERR = 1;

  /* write flash, generate 1bit error */
  {
      FLASH_EraseSector(0x10000000);
      WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x1e << 12);
      FLASH_ProgramWord(0xa5a5, 0x10000000);
      u32Data = *((__IO uint32_t *)0x10000000);
  }

  /* write flash, generate 2bit error */
  {
      FLASH_EraseSector(0x10000000);
      WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x2e << 12);
      FLASH_ProgramWord(0xa5a5, 0x10000000);
      u32Data = *((__IO uint32_t *)0x10000000);
  }

  /* read flash, generate 1bit error */
  {
      FLASH_EraseSector(0x10000000);
      WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x1e << 6);
      FLASH_ProgramWord(0xa5a5, 0x10000000);
      u32Data = *((__IO uint32_t *)0x10000000);
  }

  /* read flash, generate 2bit error */
  {
      FLASH_EraseSector(0x10000000);
      WRITE_REG(*((volatile uint32_t*)(0x40000F40)), 0x2e << 6);
      FLASH_ProgramWord(0xa5a5, 0x10000000);
      u32Data = *((__IO uint32_t *)0x10000000);
```

Example Code

```
    }

  while(1)
  {
  }
}

void MEM_IRQHandler(void)
{
    if (SYSTEM->MEMIF.bit.FLASH1ERR == 1)
    {
        printf("NMI_EVENT_FLASH_SINGLE_BIT_ERROR\n");
        SYSTEM->MEMIC.bit.FLASH1ERR = 1;
    }

    if (SYSTEM->MEMIF.bit.FLASH2ERR == 1)
    {
        printf("NMI_EVENT_FLASH_MULTI_BIT_ERROR\n");
        SYSTEM->MEMIC.bit.FLASH2ERR = 1;
    }
    SYSTEM->MEMIC.bit.MEMINT = 1;
}
```

3.4 SPC1169 系列

3.4.1 Flash、ROM、RAM ECC 功能验证

3.4.1.1 功能需求

验证 Flash、ROM、RAM 对应 ECC 功能有效

3.4.1.2 功能实现

本示例演示通过强制给 Flash、ROM、RAM0 注入单比特错误与双比特错误，产生 ECC 错误中断来验证 Flash、ROM、RAM0 具有 ECC 功能。所以验证的测试程序不能存放在 Flash、ROM、RAM0 中，需要放在 RAM1 中运行。如果需要演示检测 RAM1 ECC 错误，测试程序需要放在 RAM0 中运行（不同型号 RAM0 可用空间会有差别，参考数据手册以及技术参考手册），同时将程序中的 RAM0 ECC 错误替换为 RAM1 ECC 错误。

示例配置程序配置流程如下：

- 初始化系统时钟，UART 调试口；
- 使能产生单比特错误和多比特错误时产生中断；
- 使能操作特殊寄存器；
- 如果是 Flash ECC 功能测试，则进行扇区擦除，其它 ECC 功能测试则跳过此步骤；
- 随机出一个单比特或者双比特错误；
- 随机出一个特殊的 Flash、ROM 与 RAM0 地址（为写操作所用）；
- 如果是写入错误 ECC，则使能写错误 ECC，其它 ECC 功能测试则跳过此步骤；
- 写入数据；
- 随机出一个特殊的 Flash、ROM 与 RAM0 地址（为读操作所用）；
- 随机出一个单比特或者双比特错误；
- 如果是读取错误 ECC，则使能读错误 ECC，其它 ECC 功能测试则跳过此步骤；
- 读取数据（对于前期写入错误 ECC，因为错误 ECC 已经写入，读取同样报错）。

Example Code

```
/*****************************************************************************
 * @file     main.c
 * @brief    Main program body
 * @version  VXX.XX.XX
 * @date     XX-XXX-XXXX
 *
 * @note
 * Copyright (C) 2022 Spintrol Electronic Technology (Shanghai) Co., Ltd.. All
rights reserved.
 *
 * @attention
 * THIS SOFTWARE JUST PROVIDES CUSTOMERS WITH CODING INFORMATION REGARDING
 * THEIR PRODUCTS, WHICH AIMS AT SAVING TIME FOR THEM. SPINTROL SHALL NOT BE
 * LIABLE FOR THE USE OF THE SOFTWARE. SPINTROL DOES NOT GUARANTEE THE
```

Example Code

```
 * CORRECTNESS OF THIS SOFTWARE AND RESERVES THE RIGHT TO MODIFY THE SOFTWARE
 * WITHOUT NOTIFICATION.
 *

*******************************************************************************/


#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif

#include <stdio.h>
#include <stdlib.h>


/**
 *  @brief  Software forced memory access error options
 */
typedef enum
{
    READ_ROM_ERROR          = (1U << 0 ),   /* Error when read ROM            */
    READ_FLASH_ERROR        = (1U << 1 ),   /* Error when read FLASH          */
    READ_RAM0_ERROR         = (1U << 2 ),   /* Error when read RAM0           */
    READ_RAM1_ERROR         = (1U << 3 ),   /* Error when read RAM1           */
    READ_CAN_ERROR          = (1U << 4 ),   /* Error when read CAN            */

    WRITE_FLASH_ERROR       = (1U << 17),   /* Error when write FLASH         */
    WRITE_RAM0_ERROR        = (1U << 18),   /* Error when write RAM0          */
    WRITE_RAM1_ERROR        = (1U << 19),   /* Error when write RAM1          */
    WRITE_CAN_ERROR         = (1U << 20),   /* Error when write CAN           */

    MEMORY_ACCESS_ERROR_ALL = 0x001E001FU   /* All memory access error        */
} ENGR_MemoryAccessErrorEnum;


uint32_t u32Addr;
uint32_t u32IntCnt;
uint32_t u32IntEvt;
ErrorStatus status = SUCCESS;


/*******************************************************************************
 * @brief     Enable write access to the protected ENGR registers
 *
 * @param[in]  None
 *
 * @return     None
 *
*******************************************************************************/
#define ENGR_WALLOW()                                                          \
    WRITE_REG(*((__IO uint32_t*)(uint32_t)0x40000F70), 0x1ACCE551U)



/*******************************************************************************
 * @brief     Force bit error pattern
 *
```

Example Code

```
 * @param[in]  u8Idx  :  Bit range selection to force the memory error
 *                    0 - Bit [31: 0]
 *                    1 - Bit [63:32]
 *                    2 - Bit [95:64]
 * @param[in]  u32Pat :  Bit error pattern for the selected 32-bit
 *                    0 - Do not force errror on the bit
 *                    1 - Force error on the bit
 *
 * @return     None
 *
*******************************************************************************/
#define ENGR_ForceMemoryErrorPattern(u8Idx, u32Pat)                             \
    WRITE_REG(*((__IO uint32_t*)(uint32_t)(0x40000F58 + 4 * (u8Idx))), (u32Pat))



/*******************************************************************************
 * @brief      Force memory access error
 *
 * @param[in]  eErr: Select the memory access type to force the error
 *                 Please refer to ENGR_MemoryAccessErrorEnum
 *
 * @return     None
 *
*******************************************************************************/
#define ENGR_ForceMemoryAccessError(eErr)                                       \
    SET_BITS(*((__IO uint32_t*)(uint32_t)0x40000F64), (eErr))



void ROM_ReadErrorIntCheck(uint32_t u32Base, uint32_t u32Size,
    uint8_t u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    i;
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;

    u32IntCnt = 0U;
    ENGR_WALLOW();

    for (i = 0U; i < 3U; i++)
    {
        /* Select the test address */
        switch (rand() & 7U)
        {
            case 0U:
                u32Addr = u32Base;
                break;
            case 1U:
                u32Addr = u32Base + u32Size - 1U;
                break;
            default:
                u32Addr = ((rand() % u32Size) + u32Base);
                break;
        }

        /* Generate error data */
```

Example Code

```
        if (u8ErrorBitNum == 1U)
        {
            u32Bit0 = rand() % 39U;
            ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
        }
        if (u8ErrorBitNum == 2U)
        {
            do
            {
                u32Bit0 = rand() % 39U;
                u32Bit1 = rand() % 39U;
            } while (u32Bit0 == u32Bit1);
            if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5,
                (1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
            }
            else
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
                ENGR_ForceMemoryErrorPattern(u32Bit1 >> 5, 1U << (u32Bit1 & 31U));
            }
        }

        /* Force error event */
        ENGR_ForceMemoryAccessError(u32Access);

        /* Read from test address */
        switch (i)
        {
            case 0U:
                SYSTEM_SetNVREG1(*((__IO uint8_t *)(uint32_t)u32Addr));
                break;
            case 1U:
                u32Addr &= 0xfffffffe;
                SYSTEM_SetNVREG1(*((__IO uint16_t*)(uint32_t)u32Addr));
                break;
            case 2U:
                u32Addr &= 0xfffffffc;
                SYSTEM_SetNVREG1(*((__IO uint32_t*)(uint32_t)u32Addr));
                break;
            case 3U:
                u32Addr &= 0xfffffff8;
                SYSTEM_SetNVREG1(*((__IO uint64_t*)(uint32_t)u32Addr));
                break;
        }
        while (u32IntCnt == i);
    }
}



void RAM_AccessErrorIntCheck(uint32_t u32Base, uint32_t u32Size,
    uint8_t u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    i;
    uint32_t    u32Byte;
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;

    u32IntCnt = 0U;
```

Example Code

```
    ENGR_WALLOW();

    for (i = 0U; i < 3U; i++)
    {
        /* Select the test address */
        switch (rand() & 7U)
        {
            case 0U:
                u32Addr = u32Base;
                break;
            case 1U:
                u32Addr = u32Base + u32Size - 1U;
                break;
            default:
                u32Addr = ((rand() % u32Size) + u32Base);
                break;
        }

        switch (i)
        {
            case 0U:
                break;
            case 1U: u32Addr &= 0xfffffffeU;
                break;
            case 2U: u32Addr &= 0xfffffffcU;
                break;
            case 3U: u32Addr &= 0xfffffff8U;
                break;
        }

        /* Save test address to NVREG0 */
        SYSTEM_SetNVREG0(u32Addr);

        /* Generate error data */
        if (u8ErrorBitNum == 1U)
        {
            u32Bit0 = rand() % 13U;
            u32Bit0 += (u32Addr & 0x3) * 13U;
            u32Bit0 += (rand() % (1 << i)) * 13U;
            ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
        }
        if (u8ErrorBitNum == 2U)
        {
            do
            {
                u32Byte = rand() % (1 << i);
                u32Bit0 = rand() % 13U;
                u32Bit1 = rand() % 13U;
                u32Bit0 += (u32Addr & 0x3) * 13U;
                u32Bit1 += (u32Addr & 0x3) * 13U;
                u32Bit0 += u32Byte * 13U;
                u32Bit1 += u32Byte * 13U;
            } while (u32Bit0 == u32Bit1);
            if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5,
                (1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
            }
            else
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
```

Example Code

```
                ENGR_ForceMemoryErrorPattern(u32Bit1 >> 5, 1U << (u32Bit1 & 31U));
            }
        }

        /* Force error event */
        ENGR_ForceMemoryAccessError(u32Access);

        /* Write data to test address */
        /* Read data from test address */
        switch (i)
        {
            case 0U:
                *((__IO uint8_t *)(uint32_t)SYSTEM_GetNVREG0()) = rand();
                SYSTEM_SetNVREG0(*((__IO uint8_t *)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 1U:
                *((__IO uint16_t*)(uint32_t)SYSTEM_GetNVREG0()) = rand();
                SYSTEM_SetNVREG0(*((__IO uint16_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 2U:
                *((__IO uint32_t*)(uint32_t)SYSTEM_GetNVREG0()) = rand();
                SYSTEM_SetNVREG0(*((__IO uint32_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 3U:
                *((__IO uint64_t*)(uint32_t)SYSTEM_GetNVREG0()) = rand();
                SYSTEM_SetNVREG0(*((__IO uint64_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
        }
        while (u32IntCnt == i);
        switch (i)
        {
            case 0U:
                *((__IO uint8_t *)(uint32_t)u32Addr) = 0;
                break;
            case 1U:
                *((__IO uint16_t*)(uint32_t)u32Addr) = 0;
                break;
            case 2U:
                *((__IO uint32_t*)(uint32_t)u32Addr) = 0;
                break;
            case 3U:
                *((__IO uint64_t*)(uint32_t)u32Addr) = 0;
                break;
        }
    }
}



void FLASH_AccessErrorIntCheck(uint32_t u32Base, uint32_t u32Size,
    uint8_t u8ErrorBitNum, uint32_t u32Access)
{
    uint32_t    i;
    uint32_t    u32Bit0;
    uint32_t    u32Bit1;

    u32IntCnt = 0U;
    ENGR_WALLOW();

    /* Erase sector */
```

Example Code

```
pHWLIB->FLASHC_EraseSector(u32Base);

/* Generate error data */
if (u8ErrorBitNum == 1U)
{
    u32Bit0 = rand() % 72U;
    ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
}
if (u8ErrorBitNum == 2U)
{
    do
    {
        u32Bit0 = rand() % 72U;
        u32Bit1 = rand() % 72U;
    } while (u32Bit0 == u32Bit1);
    if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
    {
        ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5,
        (1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
    }
    else
    {
        ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
        ENGR_ForceMemoryErrorPattern(u32Bit1 >> 5, 1U << (u32Bit1 & 31U));
    }
}

/* Select the test address */
switch (rand() & 7U)
{
    case 0U:
        u32Addr = u32Base;
        break;
    case 1U:
        u32Addr = u32Base + u32Size - 1U;
        break;
    default:
        u32Addr = ((rand() % u32Size) + u32Base);
        break;
}
u32Addr = u32Addr & 0xfffffff8U;

/* Force error event */
ENGR_ForceMemoryAccessError(u32Access);

/* Flash write */
pHWLIB->FLASHC_ProgramDWord(u32Addr, rand(), rand());

/* Do not force errror on the bit */
ENGR_ForceMemoryErrorPattern(0, 0);
ENGR_ForceMemoryErrorPattern(1, 0);
ENGR_ForceMemoryErrorPattern(2, 0);

for (i = 0U; i < 3U; i++)
{
    /* Select the test address */
    switch (i)
    {
        case 0U:
            SYSTEM_SetNVREG0(u32Addr + ( rand() & 7U      ));
            break;
```

Example Code

```
            case 1U:
                SYSTEM_SetNVREG0(u32Addr + ((rand() & 3U) << 1));
                break;
            case 2U:
                SYSTEM_SetNVREG0(u32Addr + ((rand() & 1U) << 2));
                break;
        }

        /* Generate error data */
        if (u8ErrorBitNum == 1U)
        {
            u32Bit0 = rand() % 72U;
            ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
        }
        if (u8ErrorBitNum == 2U)
        {
            do
            {
                u32Bit0 = rand() % 72U;
                u32Bit1 = rand() % 72U;
            } while (u32Bit0 == u32Bit1);
            if ((u32Bit0 >> 5) == (u32Bit1 >> 5))
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5,
                (1U << (u32Bit0 & 31U)) | (1U << (u32Bit1 & 31U)));
            }
            else
            {
                ENGR_ForceMemoryErrorPattern(u32Bit0 >> 5, 1U << (u32Bit0 & 31U));
                ENGR_ForceMemoryErrorPattern(u32Bit1 >> 5, 1U << (u32Bit1 & 31U));
            }
        }

        /* Force error event */
        ENGR_ForceMemoryAccessError(u32Access);

        /* Read data from test address */
        switch (i)
        {
            case 0U:
                SYSTEM_SetNVREG0(*((__IO uint8_t *)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 1U:
                SYSTEM_SetNVREG0(*((__IO uint16_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 2U:
                SYSTEM_SetNVREG0(*((__IO uint32_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
            case 3U:
                SYSTEM_SetNVREG0(*((__IO uint64_t*)(uint32_t)SYSTEM_GetNVREG0()));
                break;
        }
        while (u32IntCnt == i);
    }
}



/*****************************************************************************
 *
```

Example Code

```
 * @brief     In this case, code and data in RAM1,
 *           test Flash, ROM, RAM0 ECC error.
 *
 * @KeyPoint   :   None
 *
 * @TestMethod :   Run code
 *
 * @TestResult :   Serial display "Test success"
 *
 ******************************************************************************/
int main(void)
{
    uint32_t u32SectorAddr;
    status = SUCCESS;

    CLOCK_InitWithRCO(100000000);

    Delay_Init();

    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Just a Sample....\n");

    /* Enable MEM_IRQHandler */
    NVIC_EnableIRQ(MEM_IRQn);

    /* Disable all Reset Event */
    SYSTEM_DisableResetEvent    (RESET_EVENT_ALL);

    /* Disable NMI and Memory Int */
    SYSTEM_DisableNonMaskableInt(NMI_EVENT_ALL);
    SYSTEM_DisableMemoryInt     (MEM_INT_ALL);

    /* Enable NMI and Memory Int */
    SYSTEM_EnableNonMaskableInt ( NMI_EVENT_ROM_MULTI_BIT_ERROR
                                | NMI_EVENT_RAM0_MULTI_BIT_ERROR
                                | NMI_EVENT_RAM1_MULTI_BIT_ERROR
                                | NMI_EVENT_FLASH_MULTI_BIT_ERROR
                                );
    SYSTEM_EnableMemoryInt      ( MEM_INT_ROM_SINGLE_BIT_ERROR
                                | MEM_INT_RAM0_SINGLE_BIT_ERROR
                                | MEM_INT_RAM1_SINGLE_BIT_ERROR
                                | MEM_INT_FLASH_SINGLE_BIT_ERROR
                                );


    u32SectorAddr = 0x1000F000;

    /* Single-bit error during read */
    u32IntEvt = MEM_INT_FLASH_SINGLE_BIT_ERROR;
    FLASH_AccessErrorIntCheck(u32SectorAddr, 0x1000, 1, READ_FLASH_ERROR);

    /* Single-bit error during write */
    u32IntEvt = MEM_INT_FLASH_SINGLE_BIT_ERROR;
    FLASH_AccessErrorIntCheck(u32SectorAddr, 0x1000, 1, WRITE_FLASH_ERROR);

    /* ROM: Read single bit error */
    u32IntEvt = MEM_INT_ROM_SINGLE_BIT_ERROR;
    ROM_ReadErrorIntCheck   (0x00001000U, 0x40, 1, READ_ROM_ERROR);
```

Example Code

```
    /* RAM1: Read single bit error */
    u32IntEvt = MEM_INT_RAM0_SINGLE_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1fffb000U, 0x1000, 1, READ_RAM0_ERROR);

    /* RAM1: Write single bit error */
    u32IntEvt = MEM_INT_RAM0_SINGLE_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1fffb000U, 0x1000, 1, WRITE_RAM0_ERROR);

    /* Two-bit error during read */
    u32IntEvt = NMI_EVENT_FLASH_MULTI_BIT_ERROR;
    FLASH_AccessErrorIntCheck(u32SectorAddr, 0x1000, 2, READ_FLASH_ERROR);

    /* Two-bit error during write */
    u32IntEvt = NMI_EVENT_FLASH_MULTI_BIT_ERROR;
    FLASH_AccessErrorIntCheck(u32SectorAddr, 0x1000, 2, WRITE_FLASH_ERROR);

    /* ROM: Read multi bit error */
    u32IntEvt = NMI_EVENT_ROM_MULTI_BIT_ERROR;
    ROM_ReadErrorIntCheck   (0x00001000U, 0x40, 2, READ_ROM_ERROR);

    /* RAM1: Read multi bit error */
    u32IntEvt = NMI_EVENT_RAM0_MULTI_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1fffb000U, 0x1000, 2, READ_RAM0_ERROR);

    /* RAM1: Write multi bit error */
    u32IntEvt = NMI_EVENT_RAM0_MULTI_BIT_ERROR;
    RAM_AccessErrorIntCheck (0x1fffb000U, 0x1000, 2, WRITE_RAM0_ERROR);

    if (status == SUCCESS)
    {
        printf("Test success\n");
    }

    while (1)
    {
    }
}



void MEM_IRQHandler(void)
{
    uint32_t u32ErrorAddr = 0;

    if (SYSTEM_GetMemoryIntFlag(MEM_INT_ALL) != (u32IntEvt | MEM_INT_GLOBAL))
    {
        status = ERROR;
        printf("MEMIF: Expect 0x%08x but read 0x%08x\n",
        u32IntEvt | MEM_INT_GLOBAL, SYSTEM_GetMemoryIntFlag(MEM_INT_ALL));
    }

    switch(u32IntEvt)
    {
        case (MEM_INT_ROM_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_ROM);
            break;
        case (MEM_INT_RAM0_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM0);
            break;
```

Example Code

```
        case (MEM_INT_RAM1_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM1);
            break;
        case (MEM_INT_FLASH_SINGLE_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_FLASH);
            break;
    }

    if (u32ErrorAddr != u32Addr)
    {
        status = ERROR;
        printf("ECC error should be detected at 0x%08x, not 0x%08x\n",
        u32Addr, u32ErrorAddr);
    }

    u32IntCnt++;

    SYSTEM_ClearMemoryInt(MEM_INT_GLOBAL | u32IntEvt);
}



void NMI_Handler(void)
{
    uint32_t u32ErrorAddr = 0;

    if (SYSTEM_GetNonMaskableIntFlag(NMI_EVENT_ALL) != (u32IntEvt |
NMI_EVENT_GLOBAL))
    {
        status = ERROR;
        printf("NMIF: Expect 0x%08x but read 0x%08x\n",
        u32IntEvt | NMI_EVENT_GLOBAL,
SYSTEM_GetNonMaskableIntFlag(NMI_EVENT_ALL));
    }

    switch(u32IntEvt)
    {
        case (NMI_EVENT_ROM_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_ROM);
            break;
        case (NMI_EVENT_RAM0_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM0);
            break;
        case (NMI_EVENT_RAM1_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_RAM1);
            break;
        case (NMI_EVENT_FLASH_MULTI_BIT_ERROR):
            u32ErrorAddr = SYSTEM_GetMemErrorAddress(MEM_FLASH);
            break;
    }

    if (u32ErrorAddr != u32Addr)
    {
        status = ERROR;
        printf("ECC error should be detected at 0x%08x, not 0x%08x\n",
        u32Addr, u32ErrorAddr);
    }

    u32IntCnt++;
```

Example Code

```
    SYSTEM_ClearNonMaskableInt(NMI_EVENT_GLOBAL | u32IntEvt);
}


/******************* Copyright (C) 2022 Spintrol Electronic Technology
(Shanghai) Co., Ltd. ***** END OF FILE ****/
```