使用M251 LDROM記錄重要系統事件

NuMicro® 32位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例代碼展示使用 M251 LDROM 記錄重要系統事件
BSP 版本	M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.004
開發平台	NuMaker-M258KG V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

以下範例程式是使用 M251 LDROM 用於記錄系統事件，包括開機次數、休眠次數、硬體錯誤 (Hard fault) 和看門狗定時器 (WDT) 重置次數。同時也會記錄硬體錯誤和看門狗定時器重置的程式執行地址。

使用者可以將這些事件記錄輸出以檢查應用程式是否有任何意外行為。而記錄硬體錯誤和看門狗定時器重置發生時的程式執行地址，有助於尋找異常的根本原因。

這個範例程式碼不需要額外的外部硬體模組。

1.1 原理

由於這個範例程式碼不需要額外的外部硬體模組，接下來的部分將著重介紹記錄區域的記憶體佈局以及如何在這個軟體專案中記錄每個系統事件。記錄區域的記憶體佈局如圖 1-1 所示。

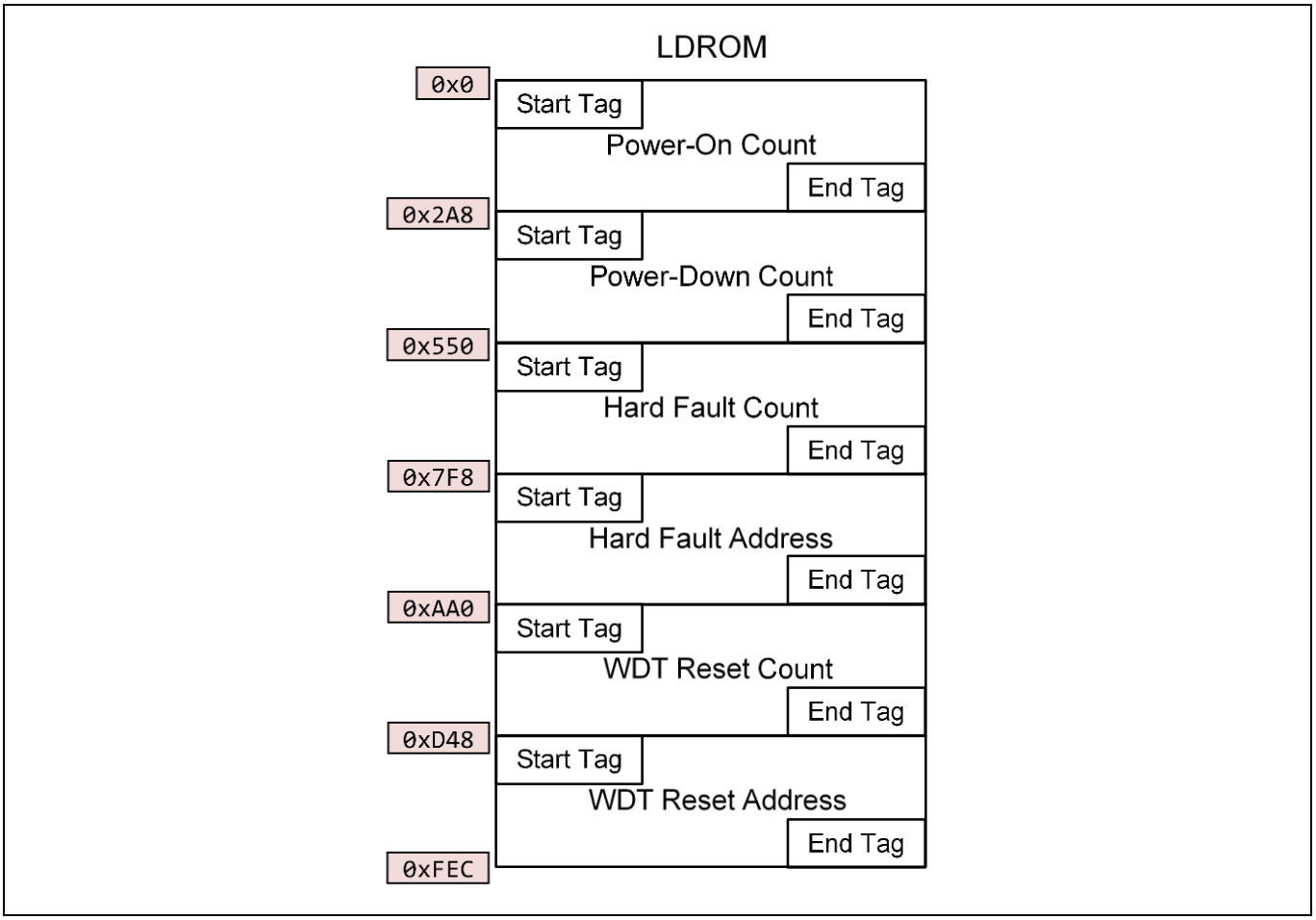

圖 1-1 日誌區域的記憶體佈局

1.2 工作流程

圖 1-2 展示系統事件記錄器範例程式碼工作流程

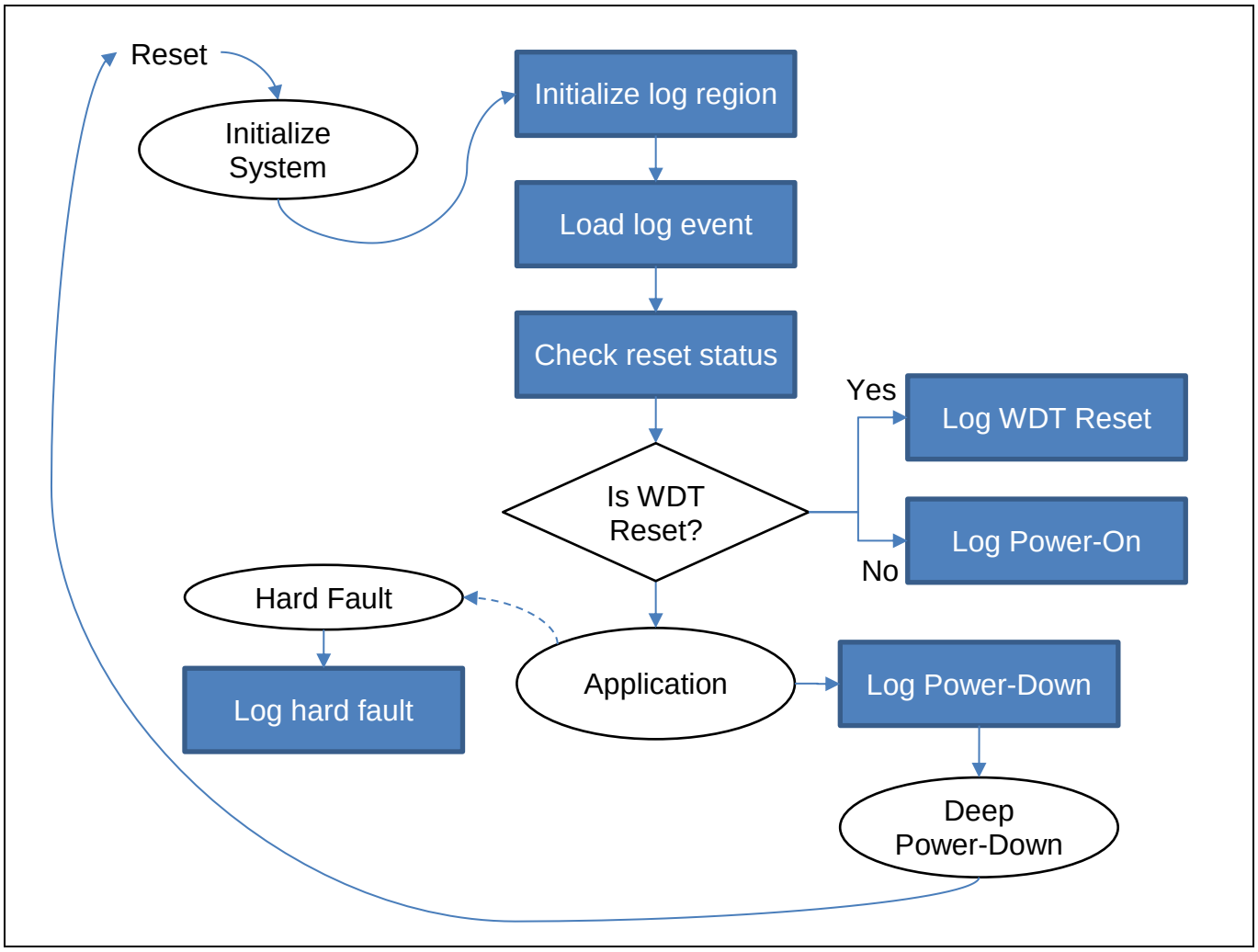

圖 1-2 系統事件記錄器工作流程

- 初始化日誌區域 (Initialize log region)

  使用者可以指定起始地址和大小來初始化日誌區域。如果在指定的區域找不到有效的標籤，則嘗試清除整塊區域並在每塊系統事件區域寫入起始和結束旗標。

  為了減少 LDROM 的寫入時間和避免資料遺失，在事件日誌記錄過程中不會進行擦除操作。當寫入事件日誌區域碰到結束旗標表示空間已滿，便無法再寫入新事件，需要重新初始化日誌區域。在範例程式中，可使用章節 1.5 執行結果的選項 [2] Clear all system events 來清除資料。

- 載入系統事件日誌 (Load log event)

  從指定的日誌區域載入系統事件計數和其發生位址。

- 檢查重置狀態旗標 (Check reset status)

  檢查系統重置狀態以識別重置來源。如果此重置是由看門狗定時器觸發的，則記錄看門狗定時器重置及其執行地址。

- 記錄開機/休眠事件 (Log Power-On/Power-Down)

  記錄開機或休眠的總次數。

  開機事件包括所有的重置來源（包括 POR、nRESET、Chip Reset、DPD Wakeup 等），但不包括 WDT Reset。

- 記錄看門狗定時器重置 (Log WDT Reset)

  看門狗定時器是一個計時器，旨在防止系統停止在未知狀態下。因此，系統需要定期重置 WDT。如果在預定的超時持續時間內未重置 WDT，WDT 將觸發 WDT 重置。

- 記錄硬體錯誤 (Log hard fault)

  在程式執行期間由於錯誤操作或無效的記憶體訪問引發的例外情況。當發生錯誤時，系統將進入錯誤例外處理程序，此時可以記錄該事件的次數和執行地址。

1.3 記錄事件發生次數

因為 Flash 寫入單位為 32 位元，而且每一位元只允許寫入兩次 0。

為了利用 LDROM 空間，這個範例程式使用 16 位元的值來記錄事件計數且最大的記錄數值為 1344（等於（0x2A8 - 0x8）* 2，參考圖 1-1）。圖 1-3 的範例顯示有 7 次事件計數。

Addr	0x00100000	0x00100004	0x00100008	0x0010000C
Data	0x00010002	0x00030004	0x00050006	0x0007FFFF
Addr	0x00100010	0x00100014	0x00100018	0x0010001C
Data	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF

圖 1-3 計數日誌範例

1.4 記錄事件發生執行位址

這個範例程式碼支援在硬體錯誤和看門狗定時器重置發生時記錄應用程式的執行地址。使用者可以檢查這些地址來尋找應用程式中可能存在的問題。

圖 1-4 顯示地址 0x0010000C 是目前的寫入標頭，最後一次儲存的執行地址為 0x0000022A，位於 0x00100008 位置。

Addr	0x00100000	0x00100004	0x00100008	0x0010000C
Data	0x0000022A	0x0000022C	0x0000022A	0xFFFFFFFF
Addr	0x00100010	0x00100014	0x00100018	0x0010001C
Data	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF

圖 1-4 位址日誌記錄範例

程式正常執行中使用者並不預期發生硬體錯誤或看門狗定時器重置，因此我們可以在例外處理程序或中斷處理程序中處理這些情況。

要在例外處理程序中擷取執行地址，使用者需要從堆疊中取得返回地址。

返回地址是被中斷程式中下一條指令的位址。這個值在例外返回時被還原到程式計數器 (PC) 上，以便中斷的程式繼續執行。

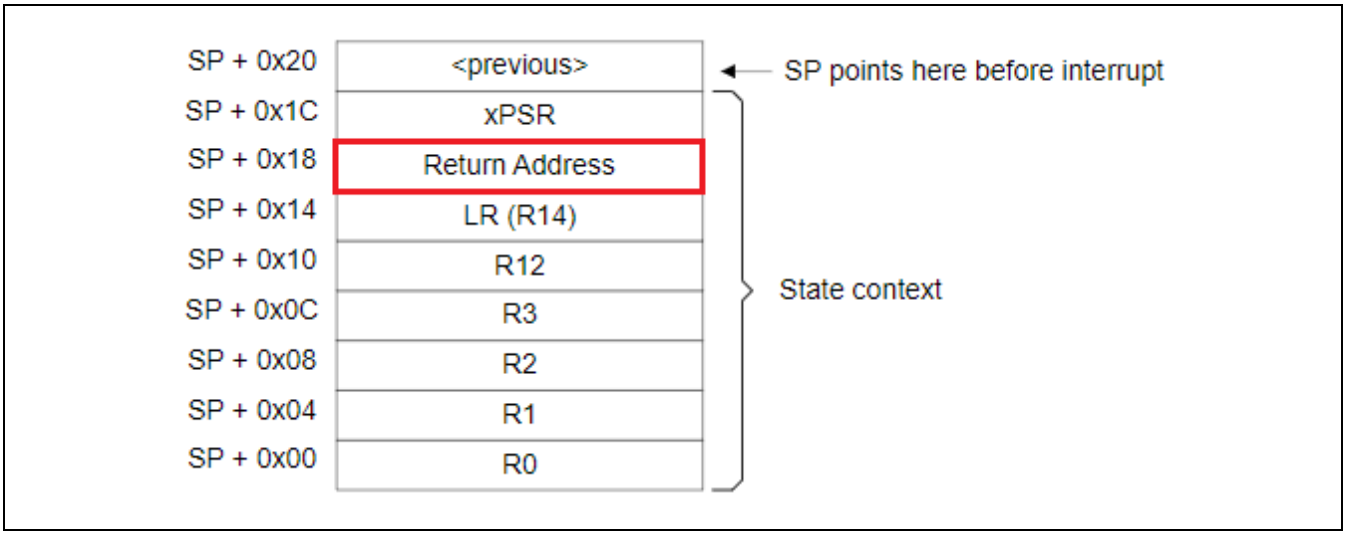

圖 1-5 堆疊框架

(Exception entry and return: https://developer.arm.com/documentation/dui1095/a/The-Cortex-M23-Processor/Exception-model/Exception-entry-and-return)

基本的堆疊框架如圖 1-5 所示，而返回位址位於 SP + 0x18 的位置。但是因為不同編譯器版本或參數可能會產生不同的堆疊框架操作，使用者必須檢查編譯器輸出的列表檔案 (KEIL\Objects\M251_System_Event_Logger.txt) 中的堆疊操作，以計算正確的返回位址偏移量。

在本範例 HardFault_Handler 中，它的堆疊指標偏移量為 6（基本偏移量）+ 2（*PUSH {r7, lr}*）+ 4（*SUB sp, sp, #0x10*）= 12 如圖 1-6。要在 C 程式碼中獲取返回位址，請參考代碼介紹中的 HardFault_Handler。

```
HardFault_Handler
        0x00000f2c:    b580        ..      PUSH    {r7,lr}
        0x00000f2e:    b084        ..      SUB     sp,sp,#0x10
        0x00000f30:    4671        qF      MOV     r1,lr
        0x00000f32:    4815        .H      LDR     r0,[pc,#84] ; [0xf88] = 0x200000e8
        0x00000f34:    6001        .`      STR     r1,[r0,#0]
        0x00000f36:    6800        .h      LDR     r0,[r0,#0]
        ...
```

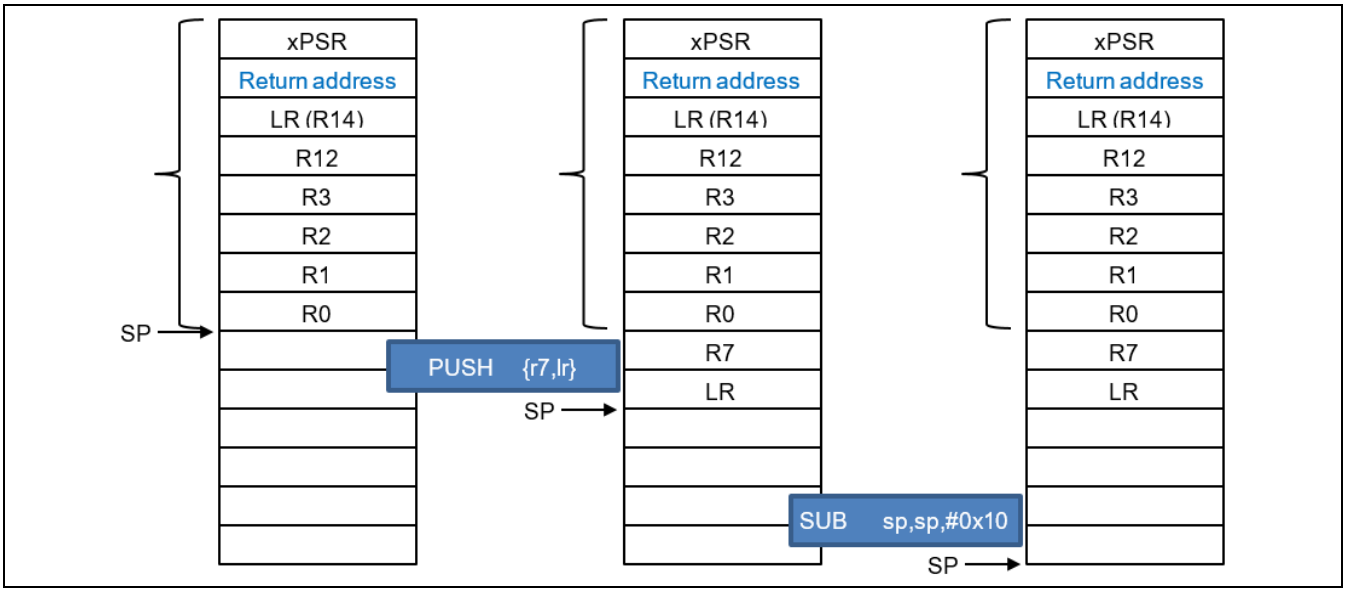

圖 1-6 HardFault_Handler 堆疊框架操作

在本範例 WDT_IRQHandler 中，它的堆疊指標偏移量為 6（基本偏移量）+ 2（*SUB sp, sp, #8*）= 8 如圖 1-7。要在 C 程式碼中獲取返回位址，請參考代碼介紹中的 WDT_IRQHandler。

```
WDT_IRQHandler
        0x00001b28:    b082        ..      SUB     sp,sp,#8
        0x00001b2a:    4671        qF      MOV     r1,lr
        0x00001b2c:    481a        .H      LDR     r0,[pc,#104] ; [0x1b98] = 0x200000e8
        0x00001b2e:    6001        .`      STR     r1,[r0,#0]
        0x00001b30:    6800        .h      LDR     r0,[r0,#0]
        ...
```

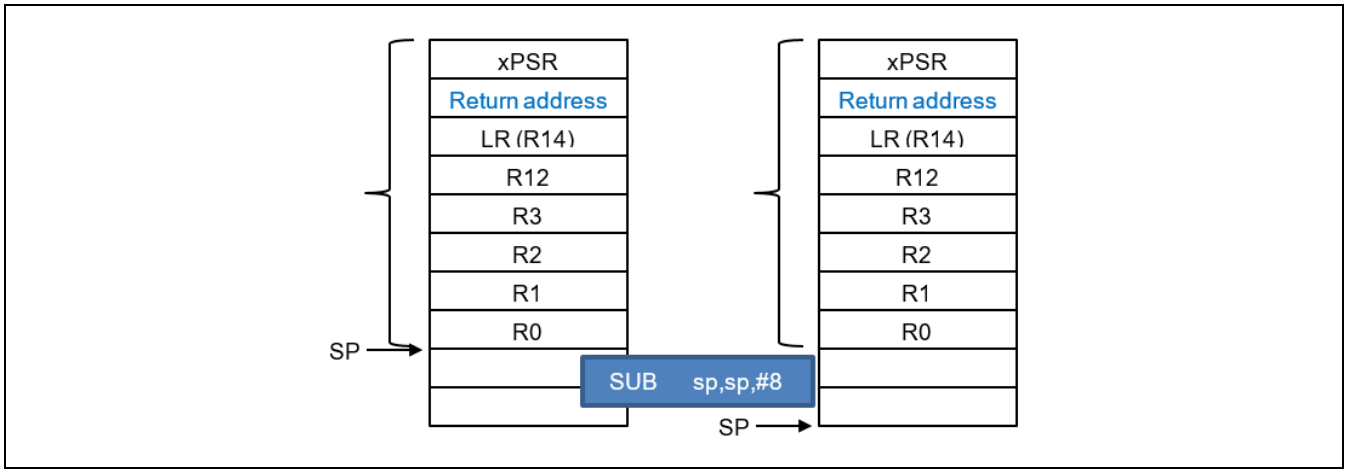

圖 1-7 WDT_IRQHandler 堆疊框架操作

1.5 執行結果

使用 UART 透過終端軟體連接並將其波特率設定為 115200。執行選單和結果將如下所示輸出。

```
================================================
              System Event Logger
================================================
================================================
  [0] Dump all system events
  [1] Dump system event summary
  [2] Clear all system events
  [3] Trigger hard fault
  [4] Trigger WDT reset
  [5] Trigger power down
  [6] Reset chip
================================================
```

- [0] Dump all system events

  輸出事件計數和所有事件發生的執行位址。

```
Select: 0
------------------------------------------------
 [ PowerOn Count   ] 2
 [ PowerDown Count ] 2
 [ HardFault Count ] 5
 [ HardFault Addr  ]
    [  5 ] 0x0000188C    [  4 ] 0x0000188C
    [  3 ] 0x0000188C    [  2 ] 0x00001886
    [  1 ] 0x0000187E
 [ WDT Reset Count ] 2
 [ WDT Reset Addr  ]
    [  2 ] 0x00000F22    [  1 ] 0x00000F14
------------------------------------------------
```

- [1] Dump system event summary

  只輸出事件計數和最後一次事件發生的執行位址。

```
Select: 1
------------------------------------------------
 [ PowerOn Count   ] 2
 [ PowerDown Count ] 2
 [ HardFault Count ] 5
 [ HardFault Addr  ] 0x0000188C
 [ WDT Reset Count ] 2
 [ WDT Reset Addr  ] 0x00000F22
------------------------------------------------
```

- [2] Clear all system events

  清除整個日誌區域並重新初始化。

- [3] Trigger hard fault

  這個範例展示三種引發硬體錯誤的方法。

  輸入 0～2 來測試記錄不同的硬體錯誤執行位址。

```
Select: 3
============================================
  [0] M32(0) = 0;
  [1] M32(FMC_LDROM_BASE + FMC_LDROM_SIZE) = 0;
  [2] M32(FMC_APROM_BASE + 0x1);
============================================
```

- [4] Trigger WDT reset

  觸發看門狗定時器重置，無訊息輸出。

- [5] Trigger power down

  進入深度省電模式，將 PC.0 從 0 變為 1 以退出省電模式。

```
Select: 5
Enter deep power down mode
  Set PC.0 from 0 to 1 to exit power down
```

- [6] Reset chip

  重置系統，無訊息輸出。

2. 代碼介紹

這個範例程式碼提供了事件日誌操作的功能函式，代碼位於 *System_Event_Logger.c* 和 *System_Event_Logger.h*。

- 常數定義

```
/* Specify default data region tag */
#define SEL_DEFAULT_TAG      0x4E554843
/*
 * TRUE:  Skip duplicated system event in the same boot cycle
 * FALSE: Log all system events in the same boot cycle
 */
#define SEL_SKIP_DUP_EVENT   (FALSE)
/* Dump how many address per line */
#define SEL_DUMP_ADDR_CNT    (2)
```

- 列舉定義

```
/* Error code */
typedef enum
{
    eSEL_ERRCODE_SUCCESS             =  0,
    eSEL_ERRCODE_INVALID_BASE_ADDR   = -1,
    eSEL_ERRCODE_ERASE_FAIL          = -2,
    eSEL_ERRCODE_WRITE_FAIL          = -3,
    eSEL_ERRCODE_INVALID_TAG         = -4,
    eSEL_ERRCODE_DATA_FULL           = -5,
    eSEL_ERRCODE_SKIP_WRITE          = -6,
    eSEL_ERRCODE_INVALID_PARAM       = -7,
} E_SEL_ERRCODE;

/* Data region type */
typedef enum
{
    eSEL_EVENT_TYPE_POWER_ON_CNT    = 0,
    eSEL_EVENT_TYPE_POWER_DOWN_CNT,
    eSEL_EVENT_TYPE_ADDR_IDX,
    eSEL_EVENT_TYPE_HARD_FAULT_CNT  = eSEL_EVENT_TYPE_ADDR_IDX,
    eSEL_EVENT_TYPE_HARD_FAULT_ADDR,
    eSEL_EVENT_TYPE_WDT_RESET_CNT,
    eSEL_EVENT_TYPE_WDT_RESET_ADDR,
    eSEL_EVENT_TYPE_CNT
} E_SEL_EVENT_TYPE;
```

- 資料結構定義

```
typedef struct
{
    uint32_t u32BaseAddr;       /* Base address to save system event    */
    uint32_t u32ByteSize;       /* Byte size to save system event       */
    uint32_t u32WriteHead;      /* Current write header                 */
    uint16_t u16DataLogCount;   /* Current data log count               */
    uint16_t bLogAddr;          /* TURE: Log address, FALSE: Log count  */
} S_SEL_DATA_REGION;

/* Total need 16 + 16 * eSEL_EVENT_TYPE_CNT = 112 bytes */
typedef struct
{
    uint32_t u32Tag;        /* Tag to identify start/end of region      */
    uint32_t u32BaseAddr;   /* Base address to save system event        */
    uint32_t u32ByteSize;   /* Byte size to save system event           */
    uint32_t u32LogMask;    /* To prevent multiple log in the same boot */
    S_SEL_DATA_REGION sDataRegion[eSEL_EVENT_TYPE_CNT];
} S_SEL_DATA_LOGGER;
```

- 全域變數

s_sDataLogger 是用於記錄日誌狀態的內部資料結構。

g_astrDataType 是每塊資料區域的常數字串陣列，用於日誌輸出。

```
static S_SEL_DATA_LOGGER s_sDataLogger = { 0 };
const char *g_astrDataType[eSEL_EVENT_TYPE_CNT] =
{
    "PowerOn Count", "PowerDown Count", "HardFault Count", "HardFault Addr", "WDT Reset
Count", "WDT Reset Addr"
};
```

- 初始化日誌區域

清除整塊日誌區域並將起始和結束標籤寫入每塊資料區域。

```
/**
  * @details    This function is used to init specified log region in APROM or LDROM to
save system event logs.
  *
  * @param[in]  u32BaseAddr Page alignment base address of log region.
  * @param[in]  u32ByteSize Page alignment byte size of log region.
  * @param[in]  u32Tag      Specify custom tag or use default tag \ref SEL_DEFAULT_TAG to
indicate start and end of different log.
  * @param[in]  bForceInit  Force to initialize log region.
  *
  * @retval     eSEL_ERRCODE_SUCCESS
  *                 Init log region successfully
  *             eSEL_ERRCODE_INVALID_BASE_ADDR
  *                 Invalid base address
  *             eSEL_ERRCODE_ERASE_FAIL
  *                 Failed to erase log region
  */
int32_t SEL_Init(uint32_t u32BaseAddr, uint32_t u32ByteSize, uint32_t u32Tag, uint32_t
bForceInit)
{
    int32_t  i;
    uint32_t u32Addr, u32DataSize;
```

```
    if ((u32BaseAddr % FMC_FLASH_PAGE_SIZE) != 0)
        return eSEL_ERRCODE_INVALID_BASE_ADDR;

    /* Region size is divided equally to data region size (4 bytes alignment) */
    u32DataSize = (u32ByteSize / eSEL_EVENT_TYPE_CNT) & ~0x3;

    /* Forced to erase whole log region to re-initialize if bForceInit == TRUE */
    if (bForceInit)
    {
        /* Erase specified flash region */
        for (u32Addr = u32BaseAddr; u32Addr < (u32BaseAddr + u32ByteSize); u32Addr +=
FMC_FLASH_PAGE_SIZE)
        {
            if (FMC_Erase(u32Addr) != 0)
            {
                return eSEL_ERRCODE_ERASE_FAIL;
            }
        }

        memset((void *)&s_sDataLogger, 0x0, sizeof(s_sDataLogger));
    }

    /* Initialize base address and size of each data log region */
    for (i = eSEL_EVENT_TYPE_POWER_ON_CNT; i < eSEL_EVENT_TYPE_CNT; i++)
    {
        s_sDataLogger.sDataRegion[i].u32ByteSize = u32DataSize;

        if (i > 0)
        {
            /* Other data region => Base address = end address of previous data region */
            s_sDataLogger.sDataRegion[i].u32BaseAddr = s_sDataLogger.sDataRegion[i -
1].u32BaseAddr + s_sDataLogger.sDataRegion[i - 1].u32ByteSize;
        }
        else
        {
            /* First data region => Base address = whole log region base address */
            s_sDataLogger.sDataRegion[i].u32BaseAddr = u32BaseAddr;
        }

        /* Check start/end tag of each data log region */
        if ((inp32(s_sDataLogger.sDataRegion[i].u32BaseAddr) != u32Tag) ||
                (inp32(s_sDataLogger.sDataRegion[i].u32BaseAddr +
s_sDataLogger.sDataRegion[i].u32ByteSize - 4) != (u32Tag + 1))
           )
        {
            /* No valid start/end tag => Write start/end tag */
            FMC_Write(s_sDataLogger.sDataRegion[i].u32BaseAddr, u32Tag);
            /* Simply set start tag + 1 as end tag */
            FMC_Write((s_sDataLogger.sDataRegion[i].u32BaseAddr +
s_sDataLogger.sDataRegion[i].u32ByteSize - 4), (u32Tag + 1));
            s_sDataLogger.sDataRegion[i].u32WriteHead =
s_sDataLogger.sDataRegion[i].u32BaseAddr + 4;
        }

        /* Set data region type to log address */
        if (i >= eSEL_EVENT_TYPE_ADDR_IDX)
            s_sDataLogger.sDataRegion[i].bLogAddr = TRUE;
```

```
    }

    s_sDataLogger.u32BaseAddr = u32BaseAddr;
    s_sDataLogger.u32ByteSize = u32ByteSize;
    s_sDataLogger.u32Tag      = u32Tag;
    s_sDataLogger.u32LogMask  = 0;

    return eSEL_ERRCODE_SUCCESS;
}
```

- 載入系統事件日誌

```
/**
  * @details    This function is used to load system event log from log region in APROM or
LDROM.
  *
  * @retval     eSEL_ERRCODE_SUCCESS:
  *                 Load
  *             eSEL_ERRCODE_INVALID_TAG
  *                 Cannot find valid tag
  */
int32_t SEL_LoadSystemEvent(void)
{
    int32_t  i;
    uint32_t u32Offset, u32Addr, u32Data;

    /* Traverse each data region to get write header and data count */
    for (i = eSEL_EVENT_TYPE_POWER_ON_CNT; i < eSEL_EVENT_TYPE_CNT; i++)
    {
        u32Offset = s_sDataLogger.sDataRegion[i].u32BaseAddr;

        /* Check base address of this data region contains start tag */
        if (inp32(u32Offset) != s_sDataLogger.u32Tag)
            return eSEL_ERRCODE_INVALID_TAG;

        /* Traverse whole data region until read end tag or 0xFFFFFFFF */
        for (u32Addr = u32Offset + 4; u32Addr < (u32Offset +
s_sDataLogger.sDataRegion[i].u32ByteSize - 4); u32Addr += 4)
        {
            u32Data = inp32(u32Addr);

            /* Reach end of data region */
            if (u32Data == (s_sDataLogger.u32Tag + 1))
                break;

            if (s_sDataLogger.sDataRegion[i].bLogAddr)
            {
                if (u32Data != 0xFFFFFFFF)
                {
                    /* Valid address log => data log count + 1 */
                    s_sDataLogger.sDataRegion[i].u16DataLogCount++;
                }
                else
                {
                    /* Set write header */
                    s_sDataLogger.sDataRegion[i].u32WriteHead = u32Addr;
                    break;
                }
            }
```

```
            else
            {
                if ((u32Data & 0xFFFF) != 0xFFFF)
                {
                    /* Valid count is in low 16 bits */
                    s_sDataLogger.sDataRegion[i].u16DataLogCount = u32Data & 0xFFFF;
                }
                else
                {
                    if ((u32Data >> 16) != 0xFFFF)
                    {
                        /* Valid count is in high 16 bits */
                        s_sDataLogger.sDataRegion[i].u16DataLogCount = u32Data >> 16;
                    }
                    /* Set write header */
                    s_sDataLogger.sDataRegion[i].u32WriteHead = u32Addr;
                    break;
                }
            }
        }
    }

    return eSEL_ERRCODE_SUCCESS;
}
```

- 輸出系統事件記錄

  使用者可修改此函式產生自定義的輸出結果。輸出結果可以參考章節 1.5 [0] Dump all system events 或[1] Dump system event summary。

```
/**
  * @details    This function is used to dump system event log from log region in APROM or
LDROM.
  *
  * @param[in]  bDumpAll To dump detail log or not
  *               TRUE:  Dump all logs
  *               FALSE: Dump log count and latest address
  *
  * @retval     eSEL_ERRCODE_SUCCESS:
  *               Success
  */
int32_t SEL_DumpSystemEvent(uint32_t bDumpAll)
{
    int32_t  i, j;

    printf("-----------------------------------------------\n");

    for (i = eSEL_EVENT_TYPE_POWER_ON_CNT; i < eSEL_EVENT_TYPE_CNT; i++)
    {
        /* Check if this region is for address and its count > 0 */
        if (s_sDataLogger.sDataRegion[i].bLogAddr &&
s_sDataLogger.sDataRegion[i].u16DataLogCount)
        {
            /* If bDumpAll == TRUE, print all log address */
            if (bDumpAll)
            {
                printf(" [ %-15s ]\n", g_astrDataType[i]);
```

```
                for (j = 0; j < s_sDataLogger.sDataRegion[i].u16DataLogCount; j++)
                {
                    if ((j > 0) && (j % SEL_DUMP_ADDR_CNT) == 0)
                        printf("\n");

                    printf("    [%3d ] 0x%08X",
(s_sDataLogger.sDataRegion[i].u16DataLogCount - j),
inp32(s_sDataLogger.sDataRegion[i].u32WriteHead - 4 * (j + 1)));
                }

                printf("\n");
            }
            else
            {
                /* If bDumpAll == FLASE, print latest log address */
                printf(" [ %-15s ] 0x%08X\n", g_astrDataType[i],
inp32(s_sDataLogger.sDataRegion[i].u32WriteHead - 4));
            }
        }
        else
        {
            /* Print data log count only */
            printf(" [ %-15s ] %d\n", g_astrDataType[i],
s_sDataLogger.sDataRegion[i].u16DataLogCount);
        }
    }

    printf("-----------------------------------------------\n");

    return eSEL_ERRCODE_SUCCESS;
}
```

- 記錄計數

  呼叫此函式以記錄指定資料事件的計數。

```
/**
  * @details    This function is used to log system event occurrence count.
  *
  * @param[in]  eEventType System event type
  *
  * @retval     eSEL_ERRCODE_SUCCESS:
  *                 Log count successfully
  *             eSEL_ERRCODE_DATA_FULL
  *                 Data region is full
  *             eSEL_ERRCODE_WRITE_FAIL
  *                 Failed to write data
  */
int32_t SEL_LogCount(E_SEL_EVENT_TYPE eEventType)
{
    uint32_t u32Offset, u32Data;
    uint16_t u16Count;

    if (s_sDataLogger.sDataRegion[eEventType].bLogAddr == TRUE)
        return eSEL_ERRCODE_INVALID_PARAM;

    /* Set SEL_SKIP_DUP_EVENT to TRUE to skip duplicated event log */
#if (SEL_SKIP_DUP_EVENT == TRUE)
```

```
    if (s_sDataLogger.u32LogMask & (1 << eEventType))
        return eSEL_ERRCODE_SKIP_WRITE;
#endif

    u16Count  = s_sDataLogger.sDataRegion[eEventType].u16DataLogCount + 1;
    u32Offset = s_sDataLogger.sDataRegion[eEventType].u32WriteHead;
    u32Data   = inp32(u32Offset);

    /* Check if log reaches end of data region */
    if (u32Data == (s_sDataLogger.u32Tag + 1))
        return eSEL_ERRCODE_DATA_FULL;

    if (u32Data == 0xFFFFFFFF)
    {
        /* Write count to high 16 bits */
        u32Data = ((u16Count << 16) | 0xFFFF);
    }
    else
    {
        /* Write count to low 16 bits */
        u32Data = (u32Data & 0xFFFF0000) | u16Count;
    }

    FMC_Write(u32Offset, u32Data);

    if (FMC_GET_FAIL_FLAG() || (g_FMC_i32ErrCode != eFMC_ERRCODE_SUCCESS))
    {
        FMC_CLR_FAIL_FLAG();
        return eSEL_ERRCODE_WRITE_FAIL;
    }

    /* Set new write header and update data log count */
    /* Check low 16 bits of this word has been updated => Switch to next word */
    if ((u32Data & 0xFFFF) != 0xFFFF)
        s_sDataLogger.sDataRegion[eEventType].u32WriteHead = u32Offset + 4;

    s_sDataLogger.sDataRegion[eEventType].u16DataLogCount = u16Count;

    /* Set SEL_SKIP_DUP_EVENT to TRUE to skip duplicated event log */
#if (SEL_SKIP_DUP_EVENT == TRUE)
    s_sDataLogger.u32LogMask |= (1 << eEventType);
#endif

    return eSEL_ERRCODE_SUCCESS;
}
```

- 記錄位址

  呼叫此函式以記錄指定資料事件的發生位址。

```
/**
  * @details    This function is used to log system event occurred address.
  *
  * @param[in]  eEventType System event type
  * @param[in]  u32PC      System event occurred address
  *
  * @retval     eSEL_ERRCODE_SUCCESS:
```

```
  *                 Success
  *             eSEL_ERRCODE_INVALID_PARAM
  *                 Specify region is not for address
  *             eSEL_ERRCODE_DATA_FULL
  *                 Data region is full
  *             eSEL_ERRCODE_SKIP_WRITE
  *                 Skip this write if u32PC == 0xFFFFFFFF or u32LogMask is set
  *             eSEL_ERRCODE_WRITE_FAIL
  *                 Failed to write data
  */
int32_t SEL_LogAddr(E_SEL_EVENT_TYPE eEventType, uint32_t u32PC)
{
    uint32_t u32Offset, u32Data;

    /* Check specified event type is for address */
    if (s_sDataLogger.sDataRegion[eEventType].bLogAddr == FALSE)
        return eSEL_ERRCODE_INVALID_PARAM;

    /* Set SEL_SKIP_DUP_EVENT to TRUE to skip duplicated event log */
#if (SEL_SKIP_DUP_EVENT == TRUE)
    if (s_sDataLogger.u32LogMask & (1 << eEventType))
        return eSEL_ERRCODE_SKIP_WRITE;
#endif

    u32Offset = s_sDataLogger.sDataRegion[eEventType].u32WriteHead;
    u32Data = inp32(u32Offset);

    /* Check not reach end of data region */
    if (u32Data == (s_sDataLogger.u32Tag + 1))
        return eSEL_ERRCODE_DATA_FULL;

    if (u32Data != 0xFFFFFFFF)
        return eSEL_ERRCODE_SKIP_WRITE;

    FMC_Write(u32Offset, u32PC);

    if (FMC_GET_FAIL_FLAG() || (g_FMC_i32ErrCode != eFMC_ERRCODE_SUCCESS))
    {
        FMC_CLR_FAIL_FLAG();
        return eSEL_ERRCODE_WRITE_FAIL;
    }

    /* Set new write header and add data log count */
    s_sDataLogger.sDataRegion[eEventType].u32WriteHead = u32Offset + 4;
    s_sDataLogger.sDataRegion[eEventType].u16DataLogCount++;

    /* Set SEL_SKIP_DUP_EVENT to TRUE to skip duplicated event log */
#if (SEL_SKIP_DUP_EVENT == TRUE)
    s_sDataLogger.u32LogMask |= (1 << eEventType);
#endif

    return eSEL_ERRCODE_SUCCESS;
}
```

- HardFault_Handler

在硬體錯誤中斷處理函式取得錯誤位址並呼叫 SEL_LogCount 和 SEL_LogAddr 記錄次數和位址。

```
void HardFault_Handler(void)
{
    __ASM volatile("mov %0, lr\n" : "=r"(g_u32LR));

    if (g_u32LR & BIT2)
        g_pu32SP = (uint32_t* )__get_PSP();
    else
        g_pu32SP = (uint32_t* )__get_MSP();
```

參考章節 1.4 了解如何從堆疊中取得返回地址

```
    /* Get information from stack */
    /* Check assembly code to get correct stack pointer offset to get return address */
    g_u32HardFaultPC = g_pu32SP[12];
    SEL_LogCount(eSEL_EVENT_TYPE_HARD_FAULT_CNT);
    SEL_LogAddr(eSEL_EVENT_TYPE_HARD_FAULT_ADDR, g_u32HardFaultPC);
    g_pu32SP[12] += 2;
}
```

- WDT_IRQHandler

WDT 是一個專門的定時器，可用於監視應用程式的執行情況，負責在系統進入未知狀態時執行 WDT 重置。

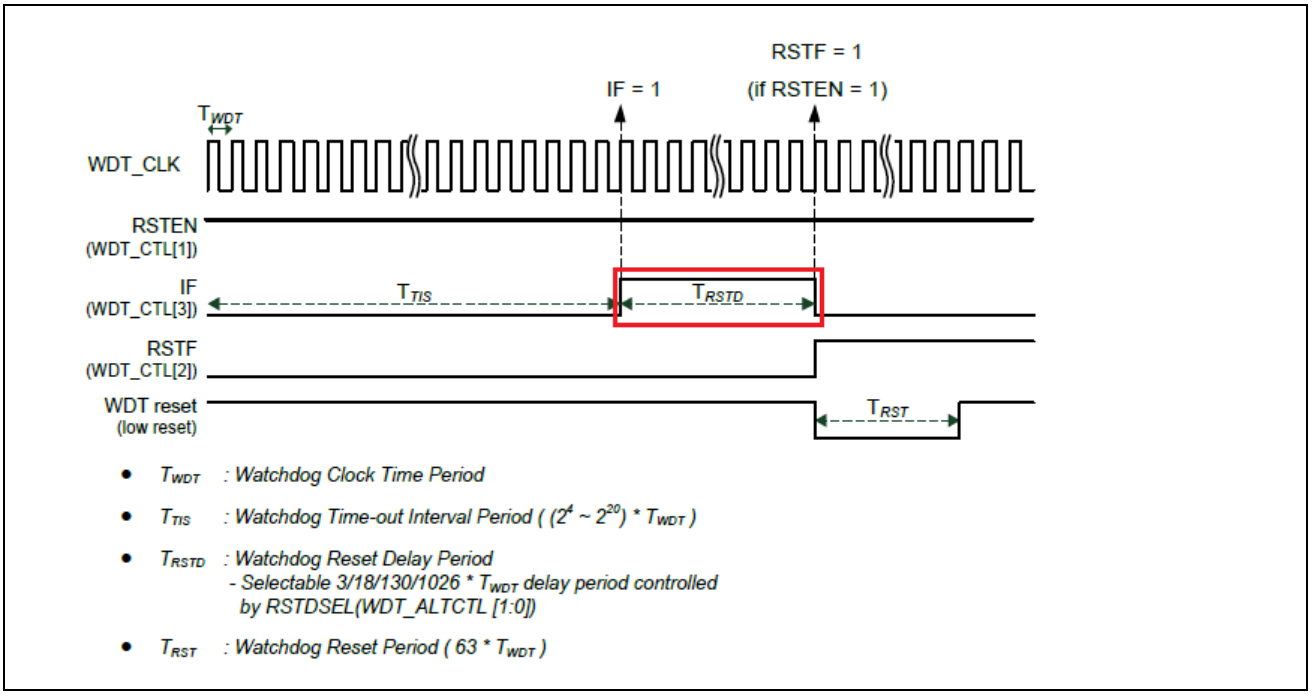

圖 2-1 WDT 重置時間時序

在圖 2-1 中：當 T_{TIS}（看門狗定時器逾時）發生時，WDT 將觸發逾時中斷（IF = 1），應用程式必須在 T_{RSTD} 週期內重置 WDT 計數器，以防止 WDT 重置系統。如果系統因某些錯誤進入未知狀態或停止，系統無法及時重置 WDT 計數器，在 T_{RSTD} 逾時後，WDT 立即觸發系統重置。

此時可能沒有足夠的時間將計數或位址記錄到 LDROM。執行位址會先保存在配置於.bss.noinit 區段的臨時變數 g_u32WDTResetPC 中，在系統重置後進行記錄。

```
volatile uint32_t g_u32WDTResetPC __attribute__((section(".bss.noinit")));
```

在某些情況下，執行位址可能無法被正確保存。例如，系統被阻塞在其他具有比 WDT_IRQHandler 更高優先級的例外處理程序或 IRQ 處理程序中。

```
void WDT_IRQHandler(void)
{
    __ASM volatile ("mov %0, lr\n" : "=r"(g_u32LR));

    if (g_u32LR & BIT2)
        g_pu32SP = (uint32_t* )__get_PSP();
    else
        g_pu32SP = (uint32_t* )__get_MSP();
```

參考章節 1.4 了解如何從堆疊中取得返回地址

```
    /* Get information from stack */
    /* Check assembly code to get correct stack pointer offset to get return address */
    g_u32WDTResetPC = g_pu32SP[8];

    if (WDT_GET_TIMEOUT_INT_FLAG() == 1)
    {
        /* For demo purpose - simply skip reset WDT counter to trigger WDT reset */
        if (g_u32TriggerWDTReset == FALSE)
            WDT_RESET_COUNTER();

        /* Clear WDT time-out interrupt flag */
        WDT_CLEAR_TIMEOUT_INT_FLAG();
    }
}
```

- 測試選單

```
void TestMenu(void)
{
    char chItem, chSubItem;

    printf("============================================\n");
    printf("  [0] Dump all system events                \n");
    printf("  [1] Dump system event summary             \n");
    printf("  [2] Clear all system events               \n");
    printf("  [3] Trigger hard fault                    \n");
    printf("  [4] Trigger WDT reset                     \n");
```

```
    printf("  [5] Trigger power down                      \n");
    printf("  [6] Reset chip                              \n");
    printf("==============================================\n");
    chItem = getchar();
    printf("Select: %c\n", chItem);

    switch(chItem - '0')
    {
        case 0:
            SEL_DumpSystemEvent(TRUE);
            break;
        case 1:
            SEL_DumpSystemEvent(FALSE);
            break;
        case 2:
            if (SEL_Init(SEL_LOG_BASE_ADDR, SEL_LOG_BYTE_SIZE, SEL_DEFAULT_TAG, TRUE) !=
eSEL_ERRCODE_SUCCESS)
                printf("Failed to init system event logger !\n");
            break;
        case 3:
            /* Test three ways to trigger hard fault */
            printf("==============================================\n");
            printf("  [0] M32(0) = 0;                             \n");
            printf("  [1] M32(FMC_LDROM_BASE + FMC_LDROM_SIZE) = 0; \n");
            printf("  [2] M32(FMC_APROM_BASE + 0x1);              \n");
            printf("==============================================\n");
            chSubItem = getchar();
            printf("Select: %c\n", chSubItem);
            switch (chSubItem - '0')
            {
                case 0:
                    M32(0) = 0;
                    break;
                case 1:
                    M32(FMC_LDROM_BASE + FMC_LDROM_SIZE) = 0;
                    break;
                default:
                    M32(FMC_APROM_BASE + 0x1);
                    break;
            }
            break;
        case 4:
            g_u32TriggerWDTReset = TRUE;
            break;
        case 5:
        {
            uint32_t u32PowerDownMode = CLK_PMUCTL_PDMSEL_DPD;

            SEL_LogCount(eSEL_EVENT_TYPE_POWER_DOWN_CNT);
            CLK_SetPowerDownMode(u32PowerDownMode);
            CLK_EnableDPDWKPin(CLK_DPDWKPIN_0, CLK_DPDWKPIN_RISING);

            printf("Enter deep power down mode\n");
            printf("  Set PC.0 from 0 to 1 to exit power down    \n");
            UART_WAIT_TX_EMPTY(UART0);
```

```
            /* Enter Power-down mode */
            CLK_PowerDown();
            printf("Wake up\n");
            break;
        }
        default:
            UART_WAIT_TX_EMPTY(UART0);
            SYS_ResetChip();
            break;
    }
}
```

- 初始化 WDT

```
void WDT_Init(void)
{
    /* Reset temporary address */
    g_u32WDTResetPC = 0xFFFFFFFF;
    /* Because of all bits can be written in WDT Control Register are write-protected;
       To program it needs to disable register protection first. */
    SYS_UnlockReg();

    /* Configure WDT settings and start WDT counting */
    WDT_Open(WDT_TIMEOUT_2POW16, WDT_RESET_DELAY_1026CLK, TRUE, FALSE);

    /* Enable WDT interrupt function */
    WDT_EnableInt();
    /* Enable WDT NVIC */
    NVIC_EnableIRQ(WDT_IRQn);
}
```

- 主程式

參考圖 1-2 描述的基本工作流程

```
int main()
{
    SYS_Init();
    /* Init UART to 115200-8n1 for print message */
    UART_Open(UART0, 115200);
    SYS_UnlockReg();
    /* Enable ISP and CONFIG/LDROM/APROM update function */
    FMC_Open();
    FMC_ENABLE_CFG_UPDATE();
    FMC_ENABLE_LD_UPDATE();
    FMC_ENABLE_AP_UPDATE();
    /* Config wake up pin 0 - PC.0 as input */
    GPIO_SetMode   (PC, BIT0, GPIO_MODE_INPUT);
    GPIO_SetPullCtl(PC, BIT0, GPIO_PUSEL_PULL_UP);

    /* Enable IAP function to access LDROM */
    if (((FMC->ISPSTS & FMC_ISPSTS_CBS_Msk) >> FMC_ISPSTS_CBS_Pos) & 1)
    {
        uint32_t au32Config[3];

        FMC_ReadConfig(au32Config, 3);
        if (au32Config[0] & 0x40)
```

```
        {
            au32Config[0] &= ~0x40;
            FMC_Erase(FMC_CONFIG_BASE);
            FMC_WriteConfig(au32Config, 3);

            printf("Wait chip reset done to enable IAP to access LDROM ...\n");
            UART_WAIT_TX_EMPTY(UART0);
            SYS_ResetChip();
            while (1)
                ;
        }
    }

    printf("\n\n");
    printf("============================================\n");
    printf("              System Event Logger           \n");
    printf("============================================\n");

    if (SEL_Init(SEL_LOG_BASE_ADDR, SEL_LOG_BYTE_SIZE, SEL_DEFAULT_TAG, FALSE) !=
eSEL_ERRCODE_SUCCESS)
        printf("Failed to init system event logger !\n");

    if (SEL_LoadSystemEvent() != eSEL_ERRCODE_SUCCESS)
        printf("Failed to load system event !\n");
```

如果該次系統重置是由看門狗定時器觸發的，則記錄計數和位址。否則將會視為正常的開機事件記錄開機次數。

```
    if ((SYS_GetResetSrc() & SYS_RSTSTS_WDTRF_Msk) && WDT_GET_RESET_FLAG())
    {
        /* Log WDT reset count and address if reset is trigger by WDT */
        SEL_LogCount(eSEL_EVENT_TYPE_WDT_RESET_CNT);
        SEL_LogAddr(eSEL_EVENT_TYPE_WDT_RESET_ADDR, g_u32WDTResetPC);
        g_u32WDTResetPC = 0xFFFFFFFF;
        SYS_ClearResetSrc(SYS_RSTSTS_WDTRF_Msk);
        WDT_CLEAR_RESET_FLAG();
    }
    else
        SEL_LogCount(eSEL_EVENT_TYPE_POWER_ON_CNT);

    WDT_Init();
    /* Generate Systick interrupt each 10 ms */
    SysTick_Config(SystemCoreClock / 100);

    while (1)
    {
        /* TestMenu is for demo purpose and can be replaced by user application code. */
        TestMenu();
    }
}
```

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M251_M252_M254_M256_M258_Series_BSP_CMSIS_V3.02.004
- IDE 版本
  - Keil uVersion 5.37

3.2 硬體需求

- 電路元件
  - NuMaker-M258KG V1.1
- 接線示意圖
  - 將 UART0 TX（PB.13）腳連接到 PC UART RX，以顯示此範例程式碼的選單和執行結果；將 UART0 RX（PB.12）腳連接到 PC UART TX，以選擇測試項目。
  - 將 Wake Up PIN0（PC.0）腳連接到 GND。

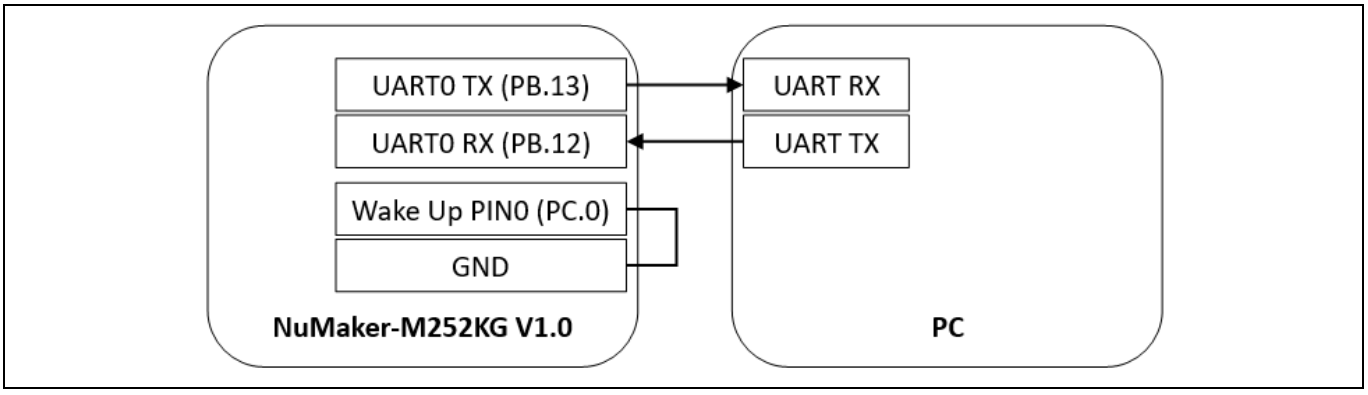

圖 3-1 接線示意圖

4. 目錄資訊

EC_M251_System_Event_Logger_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	Source file of example code

圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *M251_System_Event_Logger.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載代碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行代碼

6. 修訂紀錄

Date	Revision	Description
2023.07.15	1.00	初始發布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.