ADC 靈活的通道控制

NuMicro® 32位系列微控制器範例代碼介紹

文件資訊

應用簡述	本範例代碼使用 ADC 搭配 PDMA 達成一次觸發多個 ADC 通道，並且每個通道可以做多次 A/D 轉換。
BSP 版本	M031_Series_BSP_CMSIS_V3.05.000
開發平台	NuMaker-M032KI V1.0

1. 概述

本範例代碼使用 M031 系列的 ADC 搭配 PDMA 達成一次觸發多個 ADC 通道，並且每個通道可以做多次 A/D 轉換的功能。A/D 轉換後的所有原始資料會存放在事先定義的變數中，當一輪 ADC 轉換完成後，程式可以從變數中取出原始資料做事後分析，例如計算平均值等等。

M031 系列的 ADC 本來就支援多種運作模式。其中，Continuous Scan mode 可以一次觸發多個 ADC 通道，但每個通道只能 A/D 轉換一次，而我們需要的是每個通道要 A/D 轉換多次。舉例來說，如果要一次觸發轉換 1、2、3 三個 ADC 通道，每個通道做兩次 A/D 轉換，則 Continuous Scan mode 只能做到依序轉換通道 1、2、3、1、2、3，這和我們需要的轉換順序 1、1、2、2、3、3 不同。

本範例代碼利用 ADC Continuous Scan mode 搭配 PDMA 的自主搬動資料，計算搬動次數，和中斷功能，來達成一次觸發多個 ADC 通道，並且每個通道可以做多次 A/D 轉換的需求。

此外，為了提高範例代碼的應用範圍，程式架構設計成可以簡單的指定 ADC 通道及轉換順序，各 ADC 通道可以有不同的轉換次數，甚至可以有不同的 ADC 進階參數。細節會在後續章節中描述。

1.1 原理

要達成一次觸發多個 ADC 通道，並且每個通道可以做多次 A/D 轉換的功能，可以參考圖 1-1 的流程圖，並將整個工作切分成多個小任務，說明如下：

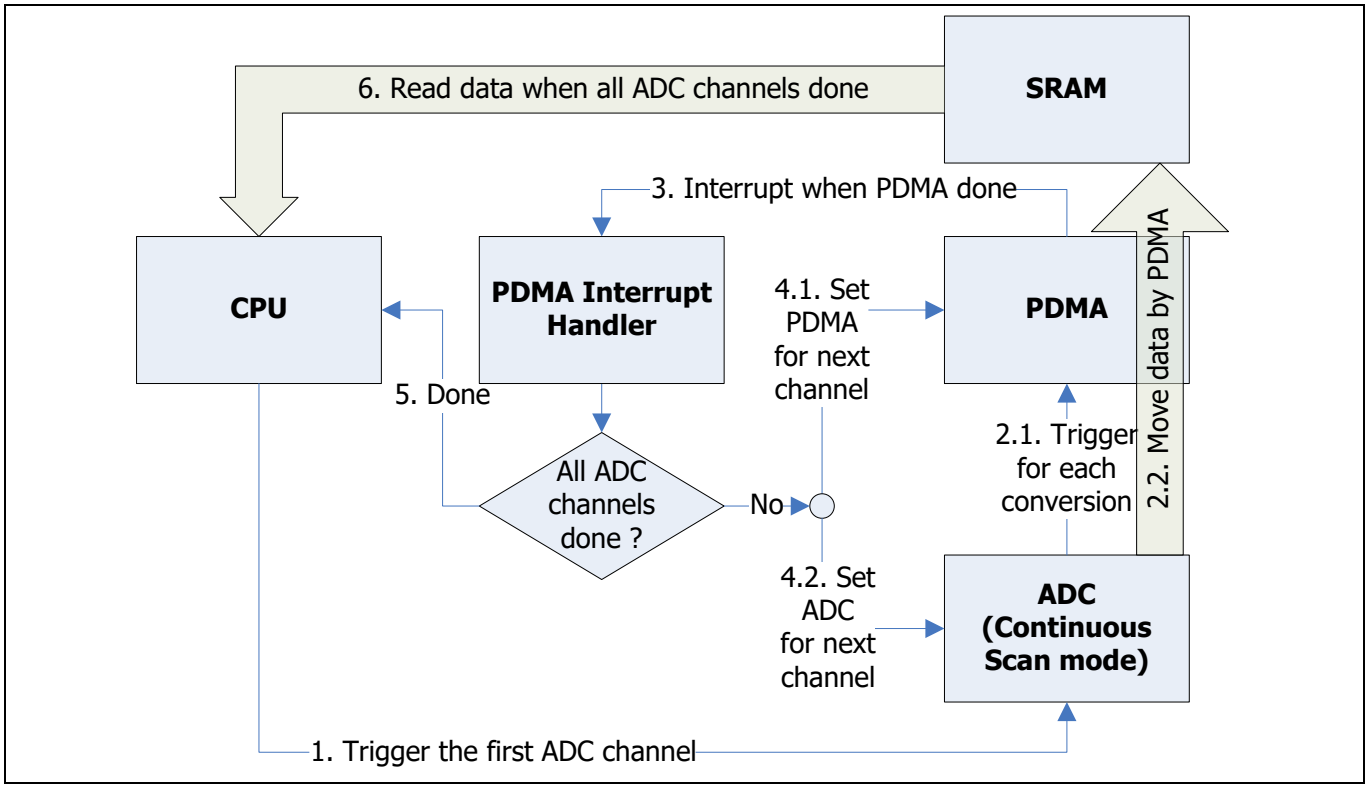

圖 1-1 程式流程圖

- 一個 ADC 通道可以 A/D 轉換多次，並把原始資料存放在指定的 SRAM 位址。

  如流程圖步驟 1 和 2，主程式流程只觸發 ADC 一次，利用 ADC Continuous Scan mode 多次轉換單一 ADC 通道，每完成 A/D 轉換一次 PDMA 就把轉換結果搬到 SRAM。

- 在 A/D 轉換達到指定次數時停止，並觸發 PDMA 中斷。

  如流程圖步驟 3，當 PDMA 搬運資料達到指定次數時，就停止搬運資料並觸發 PDMA 中斷。

- 切換到下一個 ADC 通道並重新開始 A/D 轉換。

  如流程圖步驟 4，PDMA 中斷處理程式可以修改 PDMA 和 ADC 設定，並開始轉換下一個 ADC 通道。

- 當所有 ADC 通道都 A/D 轉換完成時，停止轉換，完成一輪工作。

  如流程圖步驟 5 和 6，程式流程回到主程式，CPU 可以從 SRAM 讀取 ADC 轉換的原始資料做後續處理。

1.2 執行結果

本範例代碼的設定條件及實測環境如下：

- ADC 通道 2 轉換 6 次，PB2 接地。
- ADC 通道 1 轉換 4 次，PB1 輸入電壓約 1.82 伏特。
- ADC 通道 3 轉換 8 次，PB3 輸入電壓為 Vdd，約 3.08 伏特。

執行結果如圖 1-2 所示。

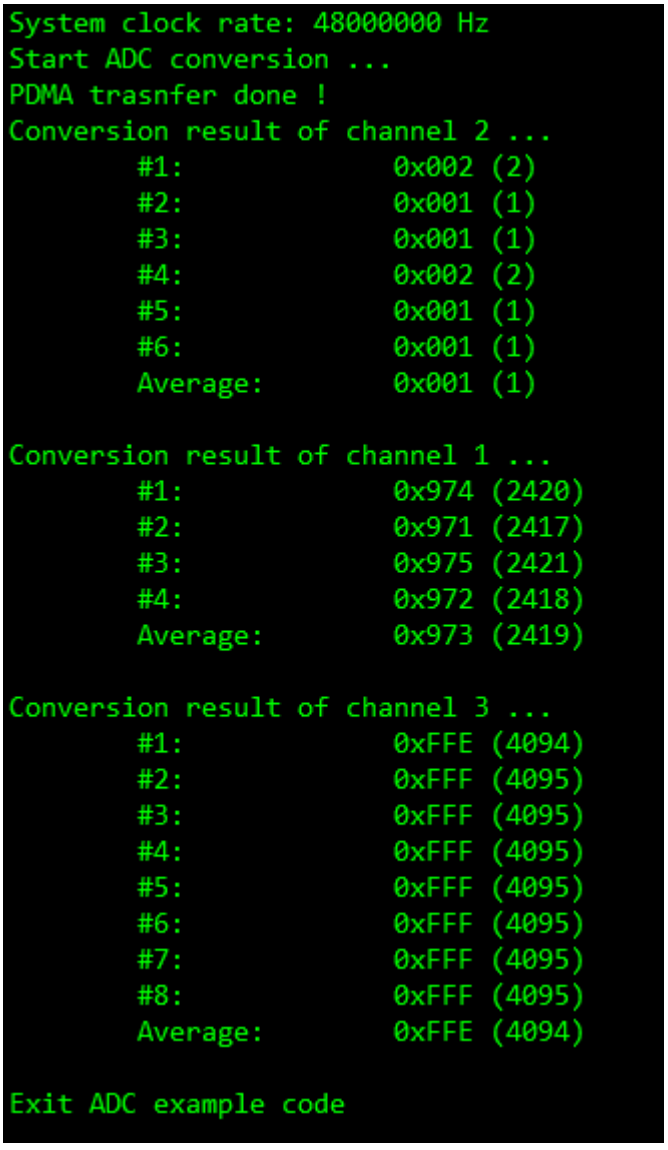

圖 1-2 執行結果

2. 代碼介紹

本範例代碼主程式會設置系統頻率為 HIRC，開啟 PDMA 運作頻率，設置 ADC 運作頻率為 HIRC/2，並將 GPIO 針腳 PB1、PB2、PB3 設置成 ADC 通道 1、通道 2、通道 3 模式，設置 GPIO 針腳 PB12、PB13 為 UART0 Rx、UART0 Tx 模式。然後執行 ADC_FunctionTest() 函數實踐一次觸發多個 ADC 通道，並且每個通道可以做多次 A/D 轉換的功能。

```
int32_t main(void)
{
    /* Init System, IP clock and multi-function I/O. */
    SYS_Init();

    /* Init UART0 for printf */
    UART0_Init();

    printf("System clock rate: %d Hz\n", SystemCoreClock);

    /* ADC function test */
    ADC_FunctionTest();

    /* Disable ADC IP clock */
    CLK_DisableModuleClock(ADC_MODULE);

    /* Disable PDMA IP clock */
    CLK_DisableModuleClock(PDMA_MODULE);

    printf("Exit ADC example code\n\n");

    while (1);
}
```

在說明 ADC 運作程式碼之前，先介紹一下如何設定 ADC 的轉換通道和轉換次數。

本範例代碼定義一個 struct 資料類型 ADC_CONF_T 來存放相關的 ADC 參數。ADC_CONF_T 的每一筆元件可以指定一個 ADC 通道以及這個 ADC 通道的轉換次數。另外，ADC_CONF_T 也可以自由增加其他的 ADC 進階參數，提供擴充程式功能的可能性。例如，本範例代碼增加了 ADC 通道的延長採樣時間，允許程式可以在不同的 ADC 通道上使用不同的延長採樣時間。

基於資料類型 ADC_CONF_F，本範例代碼定義一個全域陣列變數 g_ADC_Conf。程式會依序執行陣列中的每一個 ADC 通道，直到所有 ADC 通道都被轉換過為止。所以，根據 g_ADC_Conf 的設定值，本範例代碼的執行流程依序是：主程式流程的一次觸發就可以轉換 ADC 通道 2 六次，轉換 ADC 通道 1 四次，轉換 ADC 通道 3 八次。只要修改陣列中的值，就可以輕鬆指定 ADC 通道，A/D 轉換順序，以及 A/D 轉換次數。

```
#define ADC_MAX_CH          (3) /* The total number of channels to convert */
#define ADC_MAX_CONV_NUM    (8) /* The max number in ADC_CONF_T.number */

typedef struct
{
    uint32_t channel;   /* The specific ADC channel to convert */
    uint32_t number;    /* The number of A/D conversion for this channel */
    uint32_t extend;    /* The ADC extend sample time for this channel */
} ADC_CONF_T;

volatile ADC_CONF_T g_ADC_Conf[ADC_MAX_CH] =
{
    {2, 6, 10}, /* Convert channel 2 6 times with extend sample time 10 ADC clocks */
    {1, 4,  0}, /* Convert channel 1 4 times with extend sample time 0 ADC clock */
    {3, 8, 20}  /* Convert channel 3 8 times with extend sample time 20 ADC clocks */
};
```

為了把 ADC 轉換得到的原始資料存放在 SRAM 中，本範例代碼也定義另一個全域二維陣列變數 gu32_ADC_RawData[ADC_MAX_CH][ADC_MAX_CONV_NUM + 1]。其中，常數 ADC_MAX_CH 應該定義成要轉換的 ADC 通道的總數，在本範例代碼中就是 3。常數 ADC_MAX_CONV_NUM 應該定義成所有轉換次數中的最大值，在本範例代碼中就是 8。

需要特別說明的是，陣列大小必須是 ADC_MAX_CONV_NUM 加 1，這是因為切換 ADC 通道的期間會殘留上一輪 ADC 通道的轉換值，所以 PDMA 拿到的第一筆原始資料是屬於舊的 ADC 通道，不可以拿來使用，所以我們多定義了一個位置來存放這個殘留值，在事後分析資料時要忽略這個殘留值。

另外也要定義一個全域變數 gu32_ADC_Index 當做 g_ADC_Conf 陣列和 gu32_ADC_RawData 陣列的索引值，代表目前正在處理哪一筆設定值或哪一筆原始資料。

```
/* The raw data of ADC conversion result */
/*     Because the first conversion result may be the remaining data from
       the previous ADC channel, an extra location is declared to store it,
       but it is not actually used. */
volatile uint16_t gu32_ADC_RawData[ADC_MAX_CH][ADC_MAX_CONV_NUM + 1] = {0};

/* The index of g_ADC_Conf[] and gu32_ADC_RawData[] */
volatile uint32_t gu32_ADC_Index = 0;
```

SRAM 中各變數的具體關係可以參考下圖。

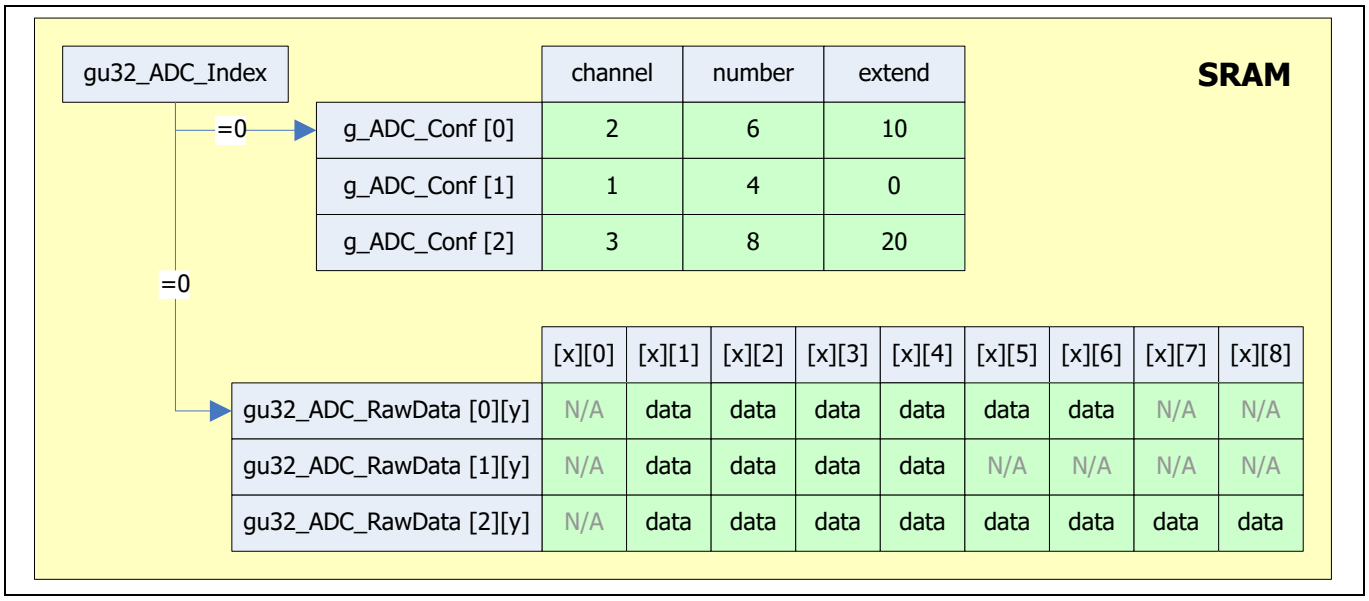

圖 2-1 SRAM 中的資料結構

接下來，可以開始說明 ADC 運作程式碼了。

如下方的程式碼，首先要設定陣列索引值 gu32_ADC_Index 為 0，讓 ADC 和 PDMA 知道接下來要依照第一筆 ADC 設定值來做 A/D 轉換。再來初始化 PDMA，這會在下個段落仔細說明。再來指定 ADC 使用 Continuous Scan mode 來轉換第一筆通道。接下來開啟 ADC 和 PDMA 之間的連動，觸發 ADC 開始做 A/D 轉換，並等待 PDMA 完成任務。PDMA 完成任務後，從 gu32_ADC_RawData 陣列變數中取得 ADC 原始資料並計算平均值，就算完成一輪工作。完整的一輪工作只使用了一次 ADC 觸發函數 ADC_START_CONV()。

```
/*---------------------------------------------------------------------*/
/* Main workflow for example code                                      */
/*---------------------------------------------------------------------*/
void ADC_FunctionTest(void)
{
    volatile uint32_t ADC_RawData_Sum[ADC_MAX_CH];
    volatile uint32_t index, number;

    /* Select the ADC configuration for the first element of g_ADC_Conf[] */
    gu32_ADC_Index = 0;

    /* Init PDMA to move data from ADC to SRAM */
    PDMA_Init();

    /* Enable ADC converter */
    ADC_POWER_ON(ADC);

    /* Wait for ADC internal power ready */
```

```
    CLK_SysTickDelay(10000);

    /* Set input mode as Single-end, and operation mode as Continuous Scan mode */
    ADC_Open(ADC, ADC_ADCR_DIFFEN_SINGLE_END, ADC_ADCR_ADMD_CONTINUOUS, (uint32_t)NULL);

    /* Select ADC input channel */
    ADC_SET_INPUT_CHANNEL(ADC, BIT0 << g_ADC_Conf[gu32_ADC_Index].channel);

    /* Set extend sampling time */
    ADC_SetExtendSampleTime(ADC, (uint32_t)NULL, g_ADC_Conf[gu32_ADC_Index].extend);

    /* ADC enable PDMA transfer */
    ADC_ENABLE_PDMA(ADC);

    gu32_PDMA_Status = PDMA_STATUS_IDLE_BUSY;

    /* Start ADC conversion */
    printf("Start ADC conversion ...\n");
    ADC_START_CONV(ADC);

    /* Wait for PDMA to complete its work */
    /* gu32_PDMA_Status will be set at PDMA_IRQHandler function */
    while (gu32_PDMA_Status == PDMA_STATUS_IDLE_BUSY);

    /* Check transfer result */
    if (gu32_PDMA_Status == PDMA_STATUS_DONE)
    {
        printf("PDMA trasnfer done !\n");
    }
    else if (gu32_PDMA_Status == PDMA_STATUS_ABORT)
    {
        printf("PDMA trasnfer abort !!\n");
    }

    /* Clear gu32_PDMA_Status software flag */
    gu32_PDMA_Status = PDMA_STATUS_IDLE_BUSY;

    /* Stop ADC conversion */
    ADC_STOP_CONV(ADC);

    /* Disable PDMA function of ADC */
    ADC_DISABLE_PDMA(ADC);
    NVIC_DisableIRQ(PDMA_IRQn);

    /* Calculate average and print ADC result except first sampling data result that
       belongs to previous channel */
    for (index = 0; index < ADC_MAX_CH; index++)
    {
        printf("Conversion result of channel %d ...\n", g_ADC_Conf[index].channel);
        ADC_RawData_Sum[index] = 0;

        for (number = 1; number < g_ADC_Conf[index].number + 1; number++)
        {
            ADC_RawData_Sum[index] += gu32_ADC_RawData[index][number];
            printf("\t#%d: \t\t0x%03X (%d)\n", number,
                   gu32_ADC_RawData[index][number], gu32_ADC_RawData[index][number]);
```

```
        }

        printf("\tAverage: \t0x%03X (%d)\n\n",
               ADC_RawData_Sum[index] / g_ADC_Conf[index].number,
               ADC_RawData_Sum[index] / g_ADC_Conf[index].number);
    }
}
```

PDMA 的初始化會設定使用 PDMA 通道 1，資料從 ADC 搬到 gu32_ADC_RawData 陣列變數，搬動的次數 (g_ADC_Conf[gu32_ADC_Index].number + 1)，允許 ADC 觸發 PDMA，並開啟 PDMA 中斷功能。PDMA 搬動次數也要多加 1，原因和前面說的一樣，是為了處理上一個 ADC 通道的殘留值。

```
/*---------------------------------------------------------------------*/
/* Configure PDMA for parameters that must be reloaded every round  */
/*---------------------------------------------------------------------*/
void ReloadPDMA(void)
{
    /* Set source address as ADC data register (no increment) and destination address as
       gu32_ADC_RawData[] array (increment) */
    PDMA_SetTransferAddr(PDMA, PDMA_CHANNEL, (uint32_t)&ADC->ADPDMA, PDMA_SAR_FIX,
    (uint32_t)gu32_ADC_RawData[gu32_ADC_Index], PDMA_DAR_INC);

    /* Transfer width is half word (16 bit) and transfer count is
       g_ADC_Conf[gu32_ADC_Index].number+1. */
    /* One more conversion is because the first conversion result belongs to the previous
       channel */
    PDMA_SetTransferCnt(PDMA, PDMA_CHANNEL, PDMA_WIDTH_16,
    g_ADC_Conf[gu32_ADC_Index].number + 1);

    /* Select PDMA request source as ADC RX */
    PDMA_SetTransferMode(PDMA, PDMA_CHANNEL, PDMA_ADC_RX, FALSE, 0);
}

/*---------------------------------------------------------------------*/
/* Configure PDMA peripheral mode from ADC to memory                */
/*---------------------------------------------------------------------*/
void PDMA_Init(void)
{
    /* Open PDMA Channel based on PDMA_CHANNEL setting*/
    PDMA_Open(PDMA, BIT0 << PDMA_CHANNEL);

    /* Configure PDMA for parameters that must be reloaded every round */
    ReloadPDMA();

    /* Set PDMA as single request type for ADC */
    PDMA_SetBurstType(PDMA, PDMA_CHANNEL, PDMA_REQ_SINGLE, 0);

    /* Enable PDMA interrupt */
    PDMA_EnableInt(PDMA, PDMA_CHANNEL, PDMA_INT_TRANS_DONE);
    NVIC_EnableIRQ(PDMA_IRQn);
}
```

當 PDMA 完成一個 ADC 通道的工作後，就要觸發中斷執行 PDMA 中斷處理程式來切換 ADC 通道。PDMA 中斷處理程式除了例行的清除中斷旗標之外，也要判斷是否所有的 ADC 通道都已經轉換完畢。如果所有的 ADC 通道都已經轉換完畢了，就可以結束一輪的工作，並把程式控制權交回給主程式；如果還有其他的 ADC 通道需要轉換，就要重新設定 PDMA 和 ADC 去處理下一個 ADC 通道，並直接觸發 ADC 轉換，不需要回到主程式中做額外的 ADC 觸發。

```
/*---------------------------------------------------------------------*/
/* PDMA interrupt handler to set ADC and PDMA for next round           */
/*---------------------------------------------------------------------*/
void PDMA_IRQHandler(void)
{
    uint32_t status;

    status = PDMA_GET_INT_STATUS(PDMA);

    if (status & PDMA_INTSTS_ABTIF_Msk)      /* PDMA abort */
    {
        if (PDMA_GET_ABORT_STS(PDMA) & (PDMA_ABTSTS_ABTIF0_Msk << PDMA_CHANNEL))
        {
            gu32_PDMA_Status = PDMA_STATUS_ABORT;
        }

        PDMA_CLR_ABORT_FLAG(PDMA, (PDMA_ABTSTS_ABTIF0_Msk << PDMA_CHANNEL));
    }
    else if (status & PDMA_INTSTS_TDIF_Msk) /* PDMA done */
    {
        if (PDMA_GET_TD_STS(PDMA) & (PDMA_TDSTS_TDIF0_Msk << PDMA_CHANNEL))
        {
            if (gu32_ADC_Index == (ADC_MAX_CH - 1))
            {
                /* All ADC channels done. Stop PDMA and trigger PDMA interrupt. */
                gu32_PDMA_Status = PDMA_STATUS_DONE;
            }
            else
            {
                ADC_STOP_CONV(ADC);
                /* Next ADC channel */
                gu32_ADC_Index++;
                /* Select ADC input channel */
                ADC_SET_INPUT_CHANNEL(ADC, BIT0 << g_ADC_Conf[gu32_ADC_Index].channel);
                /* Set extend sampling time */
                ADC_SetExtendSampleTime(ADC, (uint32_t)NULL,
                g_ADC_Conf[gu32_ADC_Index].extend);
                /* Set PDMA for new ADC channel */
                ReloadPDMA();
                ADC_START_CONV(ADC);
            }
        }

        PDMA_CLR_TD_FLAG(PDMA, (PDMA_TDSTS_TDIF0_Msk << PDMA_CHANNEL));
    }
    else
```

```
    {
        printf("Error: Unknown PDMA interrupt !!\n");
    }
}
```

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本:
  - M031_Series_BSP_CMSIS_V3.05.000.
- IDE 版本:
  - Keil uVision V4.74.

3.2 硬體需求

- 電路元件:
  - NuMaker-M032KI V1.0.

4. 目錄資訊

EC_M031_ADC_Flexible_Channel_Control_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	.
Project	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 *M031_ADC_Flexible_Channel_Control.uvproj*。
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
2024.02.16	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.