STM32 ???? ? 7 ??DMA ?????????

? ????

1. 理解 **DMA** 工作原理 - 無需 CPU 介入的高效資料轉移
2. 配置 **ADC + DMA** - 高速連續採樣
3. 配置 **UART + DMA** - 快速收發數據

? DMA ????

DMA ????

DMA (Direct Memory Access) 是一種硬體機制，允許外設直接存取記憶體，無需 CPU 介入。優點:

- 提高效率：CPU 可執行其他任務
- 降低延遲：資料轉移更快
- 減少功耗：CPU 工作量降低

STM32F446 DMA ??

特性	規格
DMA 控制器	2 個（DMA1、DMA2）
傳輸流	共 16 條流（Streams）
通道	每流 8 個通道（Channels）
最大傳輸速率	168 MB/s

DMA ????

模式	說明	應用
Normal	單次傳輸指定數量	一次性資料轉移

模式	說明	應用
Circular	循環傳輸，自動重新開始	連續採樣 /流傳輸
Mem-to-Mem	記憶體到記憶體	資料複製

?? ADC + DMA ??

?? 1?? CubeMX ??? DMA

1. 開啟之前的 ADC 專案
2. 選擇 **Analog → ADC1**
3. 在 **DMA Settings** 中點擊 **Add**
4. 設定:
   - **DMA Controller**: DMA2
   - **Stream**: Stream 0 (或其他可用流)
   - **Channel**: Channel 0 (ADC1)
   - **Priority**: High

?? 2??? DMA ??

1. 在左側找到 **DMA1** 或 **DMA2**
2. 點擊相應 Stream 進行配置:
   - **Mode**: Circular (循環模式用於連續採樣)
   - **Increment Address**: Enable (Memory)
   - **Data Width**: Word (32-bit)

?? 3?????

? ?????

main.c - ADC + DMA ????

```
/* STM32 Lesson 07 - ADC with DMA Circular Mode
 * 功能: 使用 DMA 進行高速連續 ADC 採樣
 * 難度: 中級-高級
 */
```

```c
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "usart.h"
#include "gpio.h"
#include <stdio.h>
#include <string.h>

/* 定義 */
#define ADC_BUFFER_SIZE 100

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);
static void MX_DMA_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

/* 全域變數 */
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
UART_HandleTypeDef huart1;

/* ADC 採樣緩衝區（DMA 會自動寫入） */
uint16_t adc_buffer[ADC_BUFFER_SIZE];
volatile uint8_t adc_half_complete = 0;
volatile uint8_t adc_complete = 0;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC1_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== ADC + DMA Test ===\r\n");
```

```
  /* 啟動 ADC DMA 模式 */
  HAL_ADC_Start_DMA(&hadc1, (uint32_t *)adc_buffer, ADC_BUFFER_SIZE);

  uint32_t sample_count = 0;

  while (1)
  {
    if (adc_complete)
    {
      adc_complete = 0;

      /* 計算平均值 */
      uint32_t sum = 0;
      for (uint16_t i = 0; i < ADC_BUFFER_SIZE; i++)
      {
        sum += adc_buffer[i];
      }
      uint16_t average = sum / ADC_BUFFER_SIZE;
      float voltage = (average / 4095.0f) * 3.3f;

      sample_count++;
      UART_Print("Sample %ld: ADC=%d, V=%.2f\r\n", sample_count, average, voltage);

      HAL_Delay(500);
    }
  }
}

/**
  * @brief ADC DMA 轉移完成回調
  */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_complete = 1;
  }
}
```

```
/**
  * @brief ADC DMA □□□□□
  */
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef* hadc)
{
  if (hadc->Instance == ADC1)
  {
    adc_half_complete = 1;
  }
}

/**
  * @brief UART □□□□
  */
void UART_Print(const char *format, ...)
{
  char buffer[100];
  va_list args;
  va_start(args, format);
  vsnprintf(buffer, sizeof(buffer), format, args);
  va_end(args);

  HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), HAL_MAX_DELAY);
}

/**
  * @brief □□□□□□
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
```

```
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = 2;
  RCC_OscInitStruct.PLL.PLLQ = 7;

  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
                              | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO □□□
  */
static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

/**
  * @brief DMA □□□
  */
static void MX_DMA_Init(void)
{
  __HAL_RCC_DMA2_CLK_ENABLE();

  HAL_NVIC_SetPriority(DMA2_Stream0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA2_Stream0_IRQn);
```

```
}

/**
  * @brief ADC1 □□□□DMA □□□
  */
static void MX_ADC1_Init(void)
{
  ADC_ChannelConfTypeDef sConfig = {0};

  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = ENABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = ENABLE;  /* □□ DMA □□□□ */
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;

  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  sConfig.Channel = ADC_CHANNEL_1;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_144CYCLES;

  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART1 □□□
  */
static void MX_USART1_UART_Init(void)
```

```
{
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;

  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
}

void Error_Handler(void)
{
  while(1);
}
```

? ?????

□ □□□□ : DMA 自動填充緩衝區, CPU 無需干預

? ????

???? - ? 8 ??I2C ?? + DMA

第 8 節將實現:

- **I2C** 時序與協定
- 裝置尋址與讀寫
- **I2C + DMA** 快速傳輸

□ **DMA** □□□□□□

Revision #2
Created 2026-04-01 02:06:18 UTC by TaipeiTechRacing
Updated 2026-04-06 06:23:36 UTC