STM32 ???? ? 6 ????? Timer + PWM

? ????

1. 理解定時器工作原理 - 掌握計時與計數功能
2. 掌握 **PWM** 信號產生 - 實現 LED 亮度調控
3. 實現呼吸燈效果 - 漸亮漸暗的動態視覺效果

? ?????

???????

定時器是用來計時或計數的硬體模組。STM32F446 內建 **14** 個定時器，可用於:

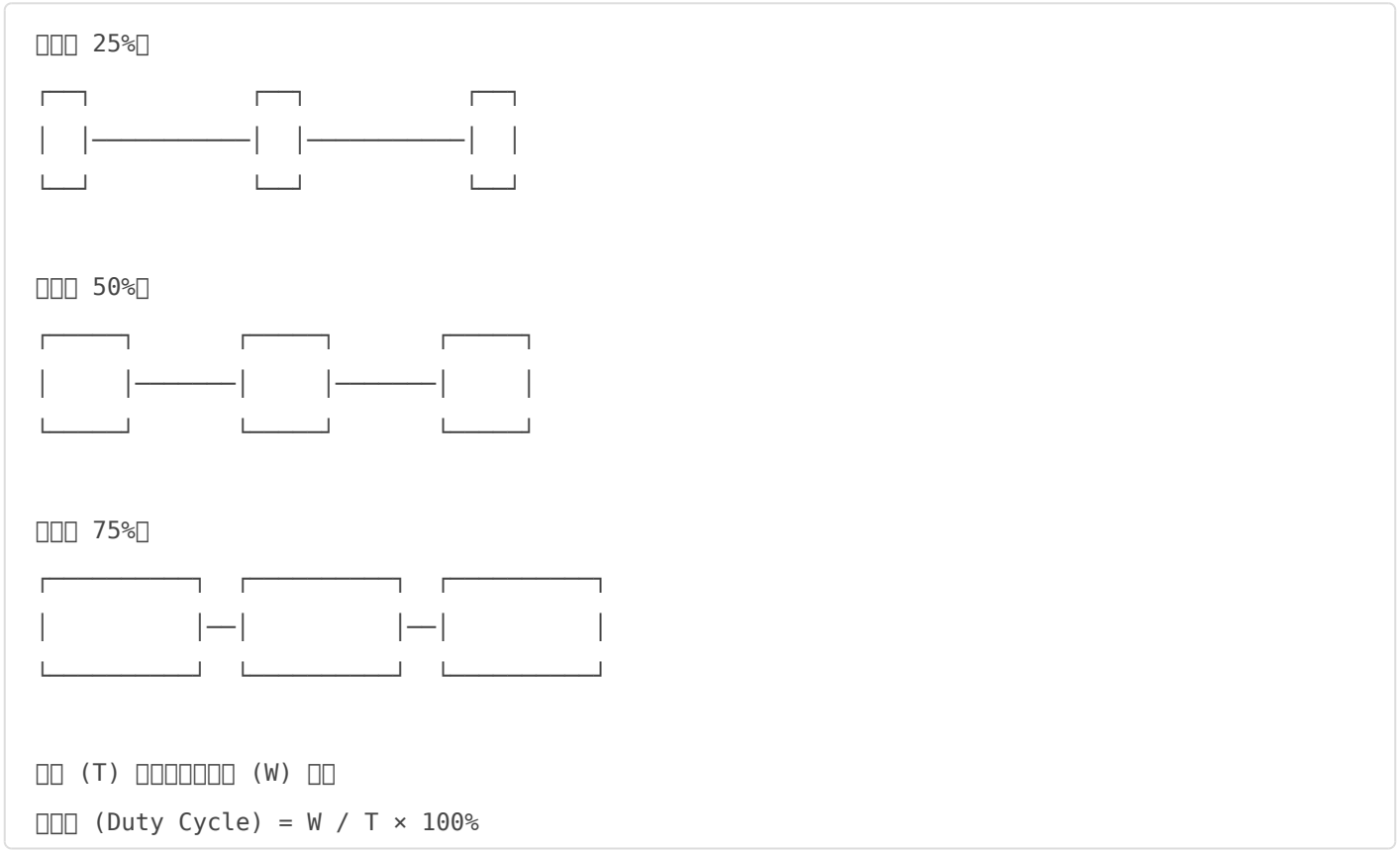
- 時間測量和延遲
- PWM 信號產生
- 外部事件計數
- 週期中斷觸發

STM32F446 ?????

定時器	數量	位寬	功能
高級定時器 (TIM1, TIM8)	2	16-bit	PWM、死區時間、重複計數
通用定時器 (TIM2-5)	4	16/32-bit	PWM、計時、中斷
基本定時器 (TIM6-7)	2	16-bit	定時中斷、DMA 觸發
低功耗定時器 (TIM9-14)	6	16-bit	低功耗應用

本課程使用 **TIM2** (通用定時器) 作為示例。

PWM ????



PWM (Pulse Width Modulation) 是透過調整脈衝寬度來控制平均功率的技術。

```
佔空比 25%:

┌──┐           ┌──┐           ┌──┐
│  │───────────│  │───────────│  │
└──┘           └──┘           └──┘


佔空比 50%:

┌─────┐       ┌─────┐       ┌─────┐
│     │───────│     │───────│     │
└─────┘       └─────┘       └─────┘


佔空比 75%:

┌──────────┐  ┌──────────┐  ┌──────────┐
│          │──│          │──│          │
└──────────┘  └──────────┘  └──────────┘


週期 (T) 固定，脈衝寬度 (W) 變化
佔空比 (Duty Cycle) = W / T × 100%
```

PWM ??

應用	說明
LED 亮度調控	改變佔空比調節亮度（0%-100%）
馬達速度控制	調整佔空比控制轉速
伺服馬達控制	脈衝寬度決定角度
功率調控	進行電源管理

?? ????

????

元件	數量
LED	1
限流電阻 (220Ω)	1

元件	數量
跳線	2

???

組件	接腳	說明
LED 正極	PA15 (TIM2_CH1)	PWM 輸出
LED 負極	GND	地線
限流電阻	LED 正負極間	保護 LED

?? CubeMX ????

?? 1??? TIM2 PWM

1. 開啟 CubeMX
2. 點擊 **Timers → TIM2**
3. 在 Pinout 圖中選擇 **PA15** 配置為 **TIM2_CH1**

?? 2??? TIM2 ??

1. 在左側選擇 **Timers → TIM2**
2. **Clock Source**: Internal Clock
3. **Channel1**: PWM Generation CH1
4. **Configuration** 中設定:
   - **Prescaler**: 839 (分頻)
   - **Counter Period**: 99 (自動重載值, ARR)
   - **Pulse**: 50 (初始佔空比 50%)

頻率計算:

```
PWM Frequency = SystemClock / ((Prescaler + 1) × ARR)
              = 168MHz / ((839 + 1) × 100)
              = 168MHz / 84000
              ≈ 2 kHz
```

?? 3????????

1. 點擊 **NVIC Settings** 標籤
2. 勾選 **TIM2 global interrupt** (用於更新佔空比)
3. 設定優先級: Preemption Priority = 3

?? 4??????

點擊 **Generate Code**

? ?????

main.c - PWM ?????

```
/* STM32 Lesson 06 - PWM Breathing LED
 * 功能: 使用 PWM 實現 LED 呼吸燈效果
 * 難度: 中級
 */

#include "main.h"
#include "tim.h"
#include "gpio.h"

/* 私有定義 */
#define PWM_MAX_VALUE    99    /* PWM 最大值 (ARR) */
#define BREATHE_SPEED    10    /* 呼吸速度 */

/* 函數宣告 */
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM2_Init(void);
void Breathing_LED(void);
void Set_PWM(uint16_t pulse_value);

/* 全域變數 */
TIM_HandleTypeDef htim2;
volatile uint16_t pwm_value = PWM_MAX_VALUE / 2;  /* 初始 50% 佔空比 */
volatile int8_t breathe_direction = 1;             /* 1: 變亮, -1: 變暗 */
```

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_TIM2_Init();

  /* 啟動 TIM2 PWM */
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  HAL_TIM_Base_Start_IT(&htim2);

  while (1)
  {
    Breathing_LED();
  }
}

/**
  * @brief 呼吸燈算法
  * 平滑改變 PWM 佔空比
  */
void Breathing_LED(void)
{
  static uint8_t counter = 0;

  counter++;

  /* 每 BREATHE_SPEED 次中斷更新一次 PWM */
  if (counter >= BREATHE_SPEED)
  {
    counter = 0;

    /* 改變 PWM 值 */
    pwm_value += breathe_direction;

    /* 邊界檢查: 反向 */
    if (pwm_value <= 0)
    {
      pwm_value = 0;
      breathe_direction = 1;  /* 開始變亮 */
```

```
    }
    else if (pwm_value >= PWM_MAX_VALUE)
    {
      pwm_value = PWM_MAX_VALUE;
      breathe_direction = -1;  /* 開始變暗 */
    }

    /* 更新 PWM */
    Set_PWM(pwm_value);
  }
}

/**
  * @brief 設定 PWM 佔空比
  * @param pulse_value: 脈衝寬度 (0 ~ PWM_MAX_VALUE)
  */
void Set_PWM(uint16_t pulse_value)
{
  __HAL_TIM_SET_COMPARE(&htim2, TIM_CHANNEL_1, pulse_value);
}

/**
  * @brief 定時器中斷回調
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (htim->Instance == TIM2)
  {
    /* 由主迴圈處理呼吸邏輯 */
  }
}

/**
  * @brief 系統時脈配置
  */
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 TIM2 □□□
```

```
  */
static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 839;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 99;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;

  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }

  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 50;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
```

```
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;

  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_TIM_MspPostInit(&htim2);
}

/**
  * @brief TIM2 MSP □□□
  */
void HAL_TIM_MspInit(TIM_HandleTypeDef* htim)
{
  if(htim->Instance == TIM2)
  {
    __HAL_RCC_TIM2_CLK_ENABLE();
    HAL_NVIC_SetPriority(TIM2_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(TIM2_IRQn);
  }
}

/**
  * @brief TIM2 MSP □□□□□GPIO □□□
  */
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* htim)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  if(htim->Instance == TIM2)
  {
    __HAL_RCC_GPIOA_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_15;
    GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    GPIO_InitStruct.Alternate = GPIO_AF1_TIM2;
    HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
  }
}

/**
  * @brief TIM2 中斷處理
  */
void TIM2_IRQHandler(void)
{
  HAL_TIM_IRQHandler(&htim2);
}

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

? ?????

????

▯ **LED 漸亮漸暗效果**:

- LED 從完全熄滅漸漸變亮至最亮
- 再漸漸變暗至完全熄滅
- 週期循環

????

問題	原因	解決方案
LED 不亮	GPIO 配置為 AF 模式失敗	檢查 `HAL_TIM_MspPostInit()` 中 Alternate 設定
PWM 頻率過高或過低	Prescaler 或 ARR 設定不當	根據需求調整計算
呼吸效果不平滑	BREATHE_SPEED 值不當	減小值使效果更平滑

? ????

???? - ? 7 ??DMA?????????

第 7 節將實現:

- **DMA** 基本概念與配置
- **ADC + DMA** 高速採樣
- **UART + DMA** 快速傳輸

□ □□□□ **PWM** □□□□□□□

Revision #2
Created 2026-04-01 02:06:09 UTC by TaipeiTechRacing
Updated 2026-04-06 06:22:55 UTC