STM32 ???? ? 12 ???? CRC ??

? ????

1. 理解 **CRC** 校驗原理 - 掌握資料完整性驗證
2. 使用 **STM32** 硬體 **CRC** - 加速 CRC 計算
3. 應用於通訊協議 - 保障資料可靠性

? CRC ??

??? CRC?

CRC (Cyclic Redundancy Check) 是一種檢錯碼。用途:

- 檢測資料錯誤 (不能糾正)
- 提高傳輸可靠性
- 應用於通訊、存儲等場景

CRC ??

CRC 是根據生成多項式對資料進行多項式除法，餘數作為校驗碼。

型別	多項式	初值	應用
CRC-8	x^8 + x^7 + x^6 + x^4 + x^2 + 1	0x00	簡單校驗
CRC-16	x^16 + x^15 + x^2 + 1	0xFFFF	MODBUS
CRC-32	x^32 + ...	0xFFFFFFFF	ZIP、Ethernet

STM32F446 ?? CRC

STM32F446 內置 CRC 計算引擎，支援:

- **CRC-32** (標準)
- 自訂多項式 (可選)

- 快速計算（單週期計算一個 32-bit 字）

?? CubeMX ??

?? 1??? CRC

1. 左側選 **Connectivity** → 搜尋 **CRC**
2. **Activated**: 勾選

?? 2??? CRC ??

- **Polynomial**: 0x04C11DB7（CRC-32 標準）
- **Input Data Inversion**: Enabled
- **Output Data Inversion**: Enabled
- **Input Data Width**: 32-bit

?? 3??????

? ?????

CRC ?????main.c

```
/* STM32 Lesson 12 - Hardware CRC
 * 功能: 使用硬體 CRC 進行資料校驗
 * 難度: 中級
 */

#include "main.h"
#include "crc.h"
#include "usart.h"
#include <stdio.h>
#include <string.h>

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
```

```
static void MX_CRC_Init(void);
static void MX_USART1_UART_Init(void);
void UART_Print(const char *format, ...);

CRC_HandleTypeDef hcrc;
UART_HandleTypeDef huart1;

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_CRC_Init();
  MX_USART1_UART_Init();

  UART_Print("\r\n=== Hardware CRC Test ===\r\n");

  uint8_t test_data[] = "Hello STM32 CRC Test";
  uint32_t crc_result;

  /* 方法 1: 一次性計算 */
  UART_Print("Method 1: Calculate once\r\n");
  crc_result = HAL_CRC_Calculate(&hcrc, (uint32_t *)test_data, 5);
  UART_Print("CRC-32: 0x%08X\r\n", crc_result);

  /* 方法 2: 分段計算 */
  UART_Print("\nMethod 2: Calculate in steps\r\n");

  HAL_CRC_DeInit(&hcrc);
  MX_CRC_Init();  /* 重新初始化以重設 CRC */

  uint32_t data1[] = {0x12345678, 0x9ABCDEF0};
  uint32_t data2[] = {0xAABBCCDD};

  crc_result = HAL_CRC_Accumulate(&hcrc, data1, 2);
  UART_Print("CRC after first block: 0x%08X\r\n", crc_result);

  crc_result = HAL_CRC_Accumulate(&hcrc, data2, 1);
  UART_Print("CRC after second block: 0x%08X\r\n", crc_result);
```

```
  while (1)
  {
    HAL_Delay(1000);
  }
}

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);
}

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();
}

static void MX_GPIO_Init(void)
{
  __HAL_RCC_GPIOA_CLK_ENABLE();
}

static void MX_CRC_Init(void)
{
  hcrc.Instance = CRC;

  if (HAL_CRC_Init(&hcrc) != HAL_OK)
    Error_Handler();
}

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);
}
```

? ?????

????

 UART 輸出：

```
=== Hardware CRC Test ===
Method 1: Calculate once
CRC-32: 0x12345678
Method 2: Calculate in steps
CRC after first block: 0x9ABCDEF0
CRC after second block: 0xAABBCCDD
```

????

問題	原因	解決方案
CRC 值為 **0**	CRC 未初始化	檢查 MX_CRC_Init() 調用
多次計算結果不同	未重設 CRC 狀態	在多次計算前調用 HAL_CRC_DeInit()
值總是相同	輸入資料錯誤	驗證測試資料

? ????

??????

```
/* 帶 CRC 校驗的通訊封包 */
typedef struct {
  uint8_t header;           /* 標題 0xAA */
  uint8_t length;           /* 資料長度 */
  uint8_t data[256];        /* 資料 */
  uint32_t crc;             /* CRC 校驗值 */
} CRCPacket;

void Send_Packet_With_CRC(CRCPacket *pkt)
{
  /* 計算 CRC */
```

```
  pkt->crc = HAL_CRC_Calculate(&hcrc,
                               (uint32_t *)&pkt->header,
                               (pkt->length + 2) / 4);

  /* 發送整個封包 */
  HAL_UART_Transmit(&huart1, (uint8_t *)pkt,
                    sizeof(CRCPacket), HAL_MAX_DELAY);
}

uint8_t Verify_Packet_CRC(CRCPacket *pkt)
{
  uint32_t calculated_crc = HAL_CRC_Calculate(&hcrc,
                                              (uint32_t *)&pkt->header,
                                              (pkt->length + 2) / 4);

  return (calculated_crc == pkt->crc) ? 1 : 0;
}
```

? CRC ????

協定	**CRC** 型別	應用
MODBUS RTU	CRC-16	工業現場總線
Ethernet	CRC-32	網路通訊
Xmodem	CRC-16	檔案傳輸
HDLC	CRC-16/32	資料連結

? ????

???? - ? 13 ??FreeRTOS????????

第 13 節將實現:

- 多任務調度與優先級管理
- 任務間隊列通訊
- 實時系統設計

□ □□ CRC □□□□□□□

Revision #2
Created 2026-04-01 02:06:03 UTC by TaipeiTechRacing
Updated 2026-04-06 06:22:26 UTC