

# MCU 性能測試，CoreMark 極簡入門教學！ (CH32V 版)

<https://bbs.21ic.com/icview-3157114-1-1.html>

本帖最後由 Litthins 於 2021-8-18 21:11 編輯

最近總結了 CoreMark 的移植方法，

正巧沁恆 RISC-V MCU 創新應用邀請賽送了 CH32V103 開發板，移植過程很簡單，分享給大家。

IDE：MounRiver Studio；MCU：CH32V103R8T6

揭開神秘面紗：CoreMark 是什麼？為什麼它可以作為 MCU 的性能指標？這個數是怎麼計算出來的？

CoreMark 是衡量嵌入式系統中微控制器性能的基準。通過包含列表處理（尋找和排序）、矩陣處理（常見的矩陣操作）、狀態機（確定輸入流是否包含有效數字）和 CRC（循環冗餘校驗）等演算法的測試給出性能評價。需要下載的軟體包是免費開放原始碼的，可以在官方網站找到：EEMBC's CoreMark®，網頁截圖如下。部分朋友無法直接下載的，文末也提供下載好的軟體包。

**EEMBC** EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM

Join ▾ Benchmarks Newsletter Press Library About

## CoreMark®

An EEMBC Benchmark

About · FAQ · Download · Scores · Submit Score · Google Group

### What is CoreMark?

CoreMark is a simple, yet sophisticated benchmark that is designed specifically to test the functionality of a processor core. Running CoreMark produces a single-number score allowing users to make quick comparisons between processors.

### Latest Certified Scores

While anyone can upload a CoreMark score, certified scores have passed the rigorous analysis of the EEMBC Certification Lab.

**Renesas Electronics RA2E1**

CoreMarks: 112

CoreMarks/MHz: 2.32

**EEMBC** CERTIFIED

### Details

EEMBC's CoreMark® is a benchmark that measures the performance of microcontrollers (MCUs) and central processing units (CPUs) used in embedded systems. Replacing the antiquated Dhrystone benchmark, Coremark contains implementations of the following algorithms: list processing (find and sort), matrix manipulation (common matrix operations), state machine

### CoreMark News

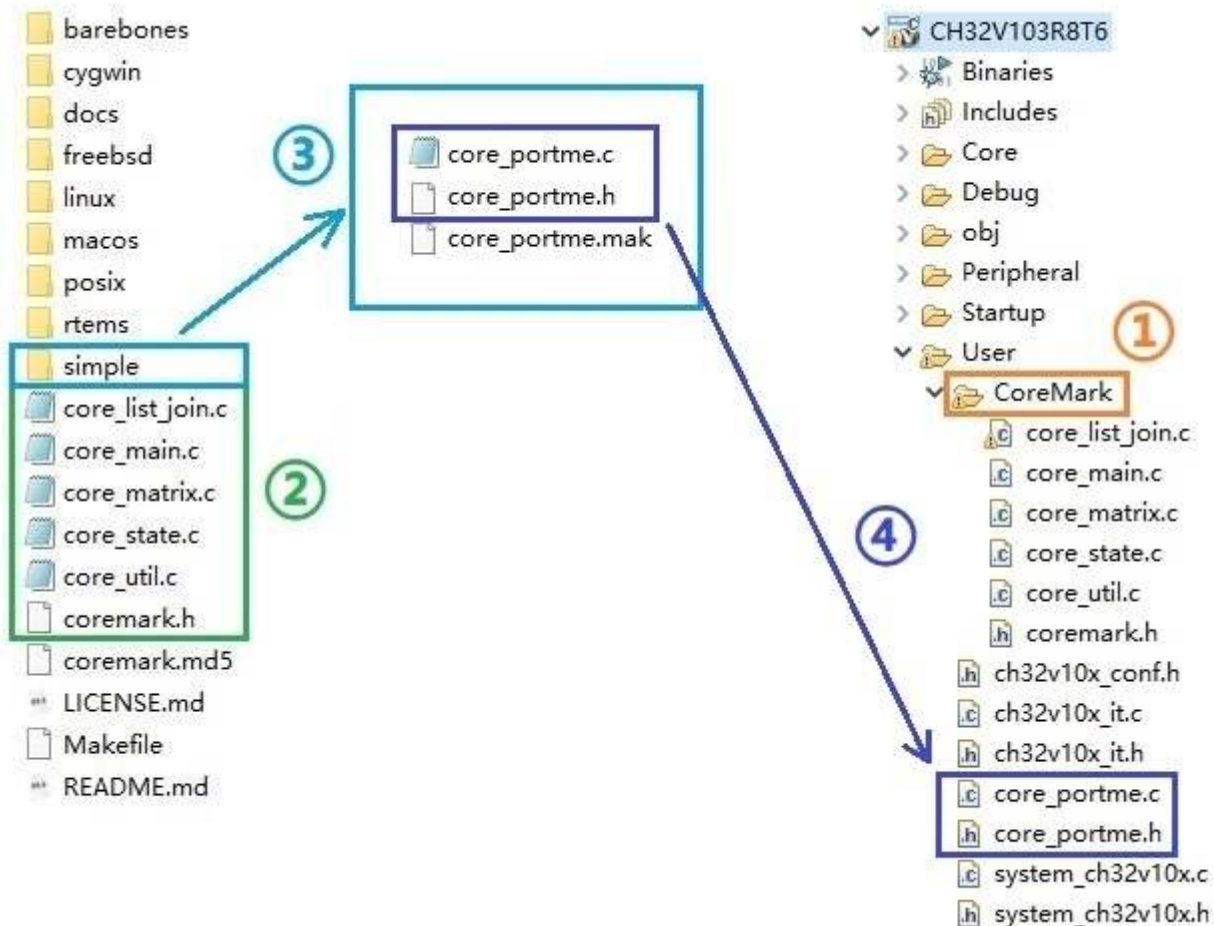
News: [EEMBC Standardizes CoreMark® Energy Measurement with New ULPMark™ Benchmark](#)

News: [STMicroelectronics New STM32L4 Microcontroller Series Devices Show Off](#)

作業不難：怎麼將 CoreMark 移植到自己的 MCU 上？運行 CoreMark 需要哪些外設支援？

運行 CoreMark，需要定時器提供計時功能，還需要向外部列印消息的手段。舉個例子，可以呼叫 `stdio.h` 中定義的 `printf()` 函數，將其重新導向到序列埠。

- **Step1:** 複製必要檔案到目標工程。現已準備好 CH32V103 的軟體工程，其檔案結構如下圖右側所示。從 CoreMark 官網獲得軟體包，軟體包解壓後檔案結構如下圖左側所示。在右側軟體工程 User 資料夾下新建 CoreMark 資料夾，將左側綠色框中 `core_list_join.c`、`core_main.c`、`core_matrix.c`、`core_state.c`、`core_util.c`、`coremark.h` 等檔案放到 CoreMark 資料夾裡。打開左側 `simple` 資料夾，將 `core_portme.c`、`core_portme.h` 複製到 User 資料夾下。



(彩色高亮標記與圖中顏色標記對應，共 4 個步驟)

- **Step2:** 由於 `core_main.c` 檔案已定義了 `main()` 函數，該 `main()` 函數執行時呼叫 `core_portme.c` 中的 `portable_init()` 函數作為 MCU 初始化介面，因此需要將 MCU 工程中原 `main.c` 檔案 MCU 初始化程式碼移動到 `portable_init()` 函數里並刪除原有 `main.c` 檔案（注意相關結構體、函數原型與函數實現要一起移動）。參考 STM32 程式碼風格，我的初始化程式碼如下：

```
void CHX_SystemClock_Config(void)
{
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
}
```

```

    RCC_APB2PeriphClockCmd(
        RCC_APB2Periph_GPIOA | RCC_APB2Periph_TIM1 | RCC_APB2Periph_USART1, ENABLE);
}

static void CHX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;

    //序列埠輸出 PA9
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
}

static void CHX_TIM1_Init(void)
{
    TIM_TimeBaseInitTypeDef TIM1_TimeBaseInitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    TIM1_TimeBaseInitStructure.TIM_Period = 10 - 1;
    TIM1_TimeBaseInitStructure.TIM_Prescaler = 7200 - 1;
    TIM1_TimeBaseInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM1_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM1, &TIM1_TimeBaseInitStructure);

    TIM_Cmd(TIM1, ENABLE);
    TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);

    NVIC_InitStructure.NVIC_IRQChannel = TIM1_UP_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 2;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

static void CHX_USART_Init(void)
{
    USART_InitTypeDef USART_InitStructure;

```

```

USART_InitStructure.USART_BaudRate = 115200;
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl =
    USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Tx;

USART_Init(USART1, &USART_InitStructure);
USART_Cmd(USART1, ENABLE);
}

```

- **Step3**：CoreMark 的分數最終表示為 **Iterations/Sec**，也就是每秒迭代數，而 **Sec** 和系統 **Ticks** 相關。用過 RTOS 的朋友應該對這個概念很熟悉，考慮文字不太好解釋，直接看 [core\\_portme.c](#) 裡這個宏定義。

```
#define EE_TICKS_PER_SEC 1000
```

- 這裡我定義每秒 **1000** 個 **Tick**，每個 **Tick** 時長 **1ms**，對應定時器每 **1ms** 觸發一次中斷。使用一個計數變數，定時器進入中斷一次，該變數值+1；對該變數值/**1000** 即可求得定時器執行階段長，也就是上文的 **Sec**。所以 **EE\_TICKS\_PER\_SEC** 並非一定要設定為 **1000**，和定時器中斷頻率對應即可。與定時相關的函數有以下三個，

```

void start_time(void);           //初始化計時器；
void stop_time(void);           //停止計時器；
CORE_TICKS get_time(void);       //獲取計時器的計數值

```

- 在 **start\_time()** 裡實現定時器啟動功能，在 **stop\_time()** 裡實現定時器停止功能，在 **get\_time()** 中獲取中斷計數值。資料類型“**CORE\_TICKS**”實際上就是“**unsigned long**”。為方便操作，推薦將計數變數設定為全域變數，這樣可以通過 **extern** 關鍵字直接訪問。此處以 **TIM1** 為例，程式碼如下：

```

//計數變數
unsigned long time_ms_ticks = 0;

//定時器啟動
void start_time(void)
{

```

```

    TIM_Cmd(TIM1, ENABLE);
    TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);
}

//定時器停止
void stop_time(void)
{
    TIM_ITConfig(TIM1, TIM_IT_Update, DISABLE);
    TIM_Cmd(TIM1, DISABLE);
}

//獲取中斷計數值
CORE_TICKS get_time(void)
{
    return time_ms_ticks;
}

//中斷處理函數
extern unsigned long time_ms_ticks;

void TIM1_UP_IRQHandler(void)
{
    time_ms_ticks++;
    if (TIM_GetITStatus(TIM1, TIM_IT_Update) != RESET)
    {
        TIM_ClearITPendingBit(TIM1, TIM_IT_Update);
    }
}

```

- 註釋部分不使用的程式碼。以下程式碼位於 `core_portme.c` 中，需要註釋掉：

```

#define NSECS_PER_SEC      CLOCKS_PER_SEC
#define CORETIMETYPE       clock_t
#define GETMYTIME(_t)      (*_t = clock())
#define MYTIMEDIFF(fin, ini) ((fin) - (ini))
#define TIMER_RES_DIVIDER  1
#define SAMPLE_TIME_IMPLEMENTATION 1

//static CORETIMETYPE start_time_val, stop_time_val;

```

- CoreMark 要求的最短測試時間為 10s，若測試時間低於 10s 則會報錯，見下圖：

```
CoreMark Start!
2K performance run parameters for coremark.
CoreMark Size      : 666
Total ticks        : 4249
Total time (secs)  : 4.249000
Iterations/Sec     : 117.674747
ERROR! Must execute for at least 10 secs for a valid result!
Iterations         : 500
Compiler version   : GCC8.2.0
Compiler flags     : -Ofast -g3
Memory location    : STACK
seedcrc           : 0xe9f5
[0]crclist        : 0xe714
[0]crcmatrix      : 0x1fd7
[0]crcstate       : 0x8e3a
[0]crcfinal       : 0xa14c
Errors detected
```

- 為獲得有效的測試結果，需修改 `core_portme.c` 中關於 `ITERATIONS` 的設定，官方案式碼中 `ITERATIONS` 沒有定義：

```
volatile ee_s32 seed4_volatile = ITERATIONS;
```

- 此處使用 CH32V103，主頻 72MHz。經測試 `ITERATIONS` 修改為 1200 左右即可，

```
volatile ee_s32 seed4_volatile = 1200;
```

- Step4：**列印測試結果時，編譯器最佳化等級和偵錯等級也可以列印出來。這類資訊可在 `core_portme.h` 中通過宏 `COMPILER_FLAGS` 修改。這裡我使用 `-Ofast` 最佳化，偵錯等級 `-g3`，修改如下：

```
#ifndef COMPILER_FLAGS
#define COMPILER_FLAGS "-Ofast -g3"
#endif
```

- 重新導向 `printf` 到序列埠，可參考以下程式碼（從 CH32V 官方例程 `debug.c` 中獲取），需要組態 IDE 新增 `float` 類型支援：

```
/* *****
 * Function Name   : _write
 * Description    : Support Printf Function
 * Input          : *buf: UART send Data.
 *                : size: Data length
 * Return         : size: Data length
 */
```



```

*****/
int _write(int fd, char *buf, int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET)
            ;
        USART_SendData(USART1, *buf++);
    }
    return size;
}

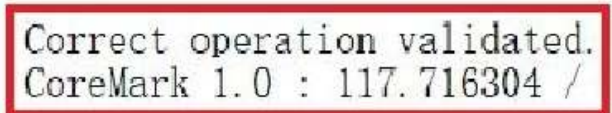
```

- **Step5 :** 移植完成！編譯程序下載運行，得到跑分結果：

```

CoreMark Start!
2K performance run parameters for coremark.
CoreMark Size      : 666
Total ticks        : 10194
Total time (secs)  : 10.194000
Iterations/Sec     : 117.716304
Iterations         : 1200
Compiler version   : GCC8.2.0
Compiler flags     : -Ofast -g3
Memory location    : STACK
seedcrc            : 0xe9f5
[0]crclist         : 0xe714
[0]crcmatrix       : 0x1fd7
[0]crcstate        : 0x8e3a
[0]crcfinal        : 0x988c
Correct operation validated. See README.md for run and reporting rules.
CoreMark 1.0 : 117.716304 / GCC8.2.0 -Ofast -g3 / STACK

```



CH32V103R8，72MHz，117.7 分！

STM32G071RB，64MHz，跑分為 108.9 分，兩款 MCU 分數接近。