1.23 不需要修改 BOOT 引腳從 SRAM 仿真

https://blog.csdn.net/oDuanYanGuHong/article/details/116239607

斷雁孤鴻 2021-04-28 18:03:29 36 收藏 2
分類專欄： Cubemx 文章標簽： STM32 SRAM 仿真 SRAM 仿真 keil SRAM 仿真

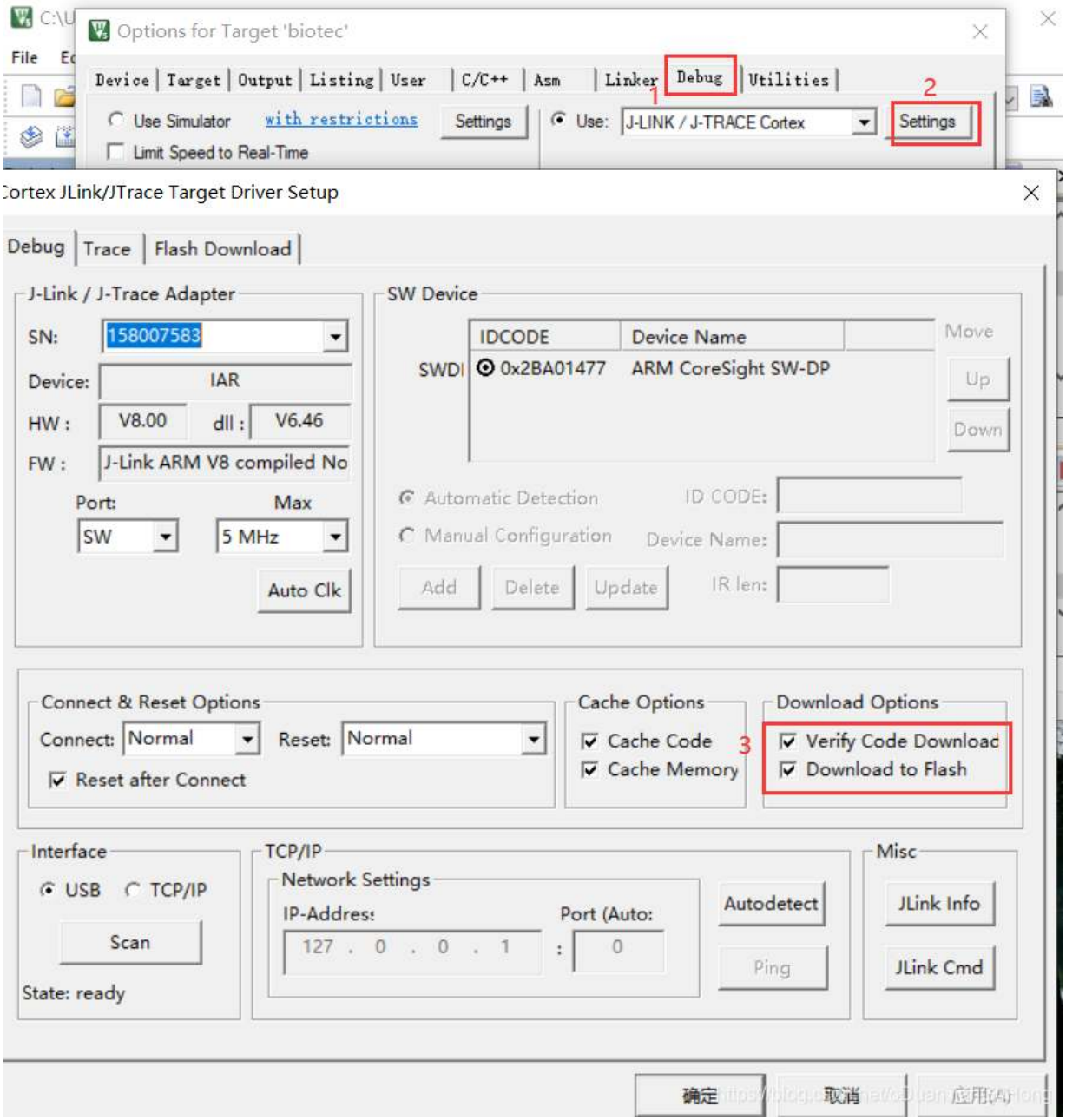
文章目錄

1、代碼從 SRAM 仿真配置。

前面文章代碼可以放到 SRAM 中來跑，但是這樣就非常不方便我們去仿真了。但是 keil 支持從 SRAM 仿真。

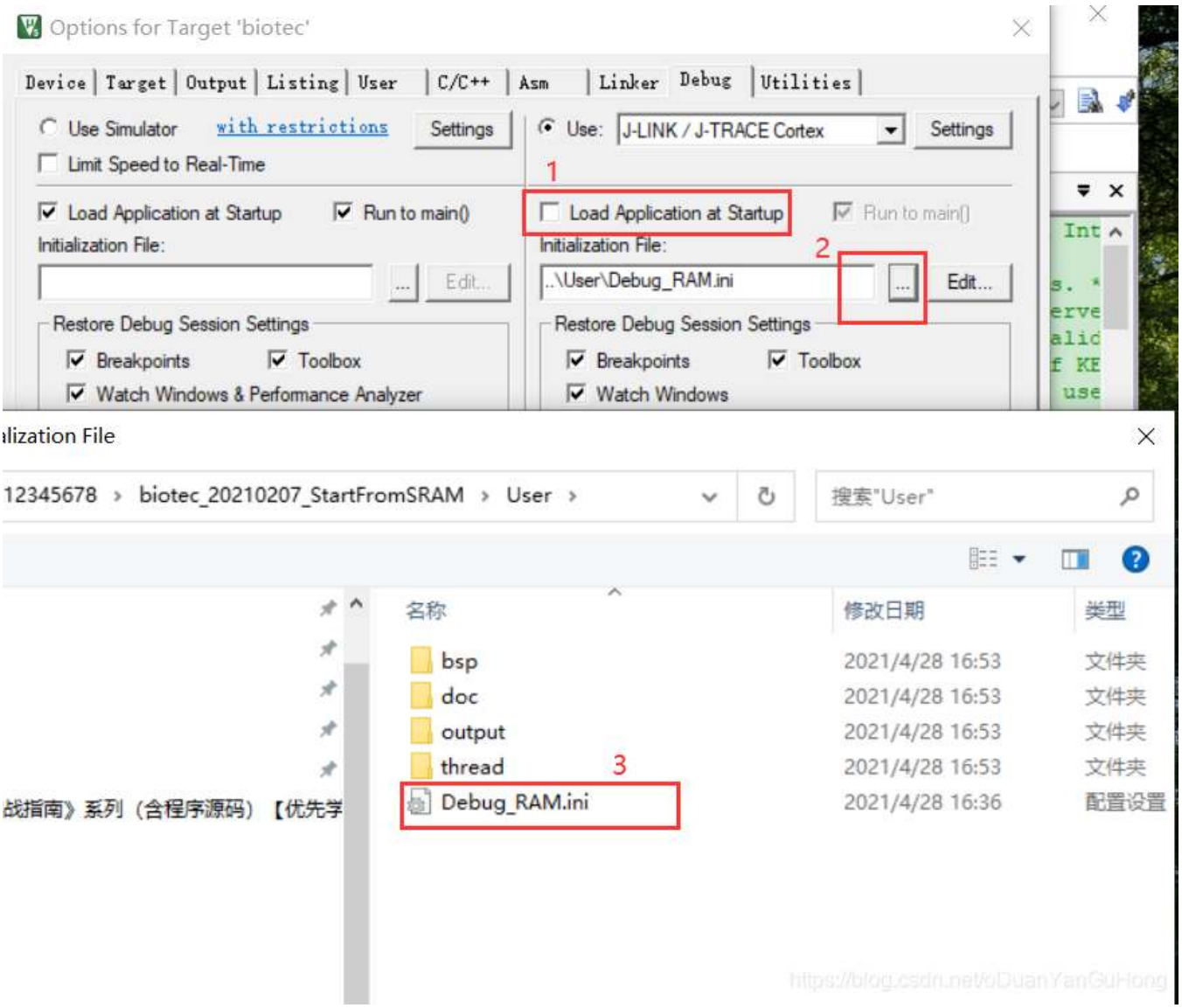
假如您使用的硬件平台中 BOOT0 和 BOOT1 引腳電平已被固定，設置為內部 FLASH 啟動，不方便改成 SRAM 方式，可以使用如下方法配置調試選項實現在 SRAM 調試：

（1）、勾選“Verify Code Download”及“Download to FLASH”選項。



（2）見下圖，在“Options for Target->Debug”對話框中取消勾選“Load Application at startup”選項。點擊“Initialization File”文本框右側的文件瀏覽

按鈕，在彈出的對話框中新建一個名為“Debug_RAM.ini”的文件；

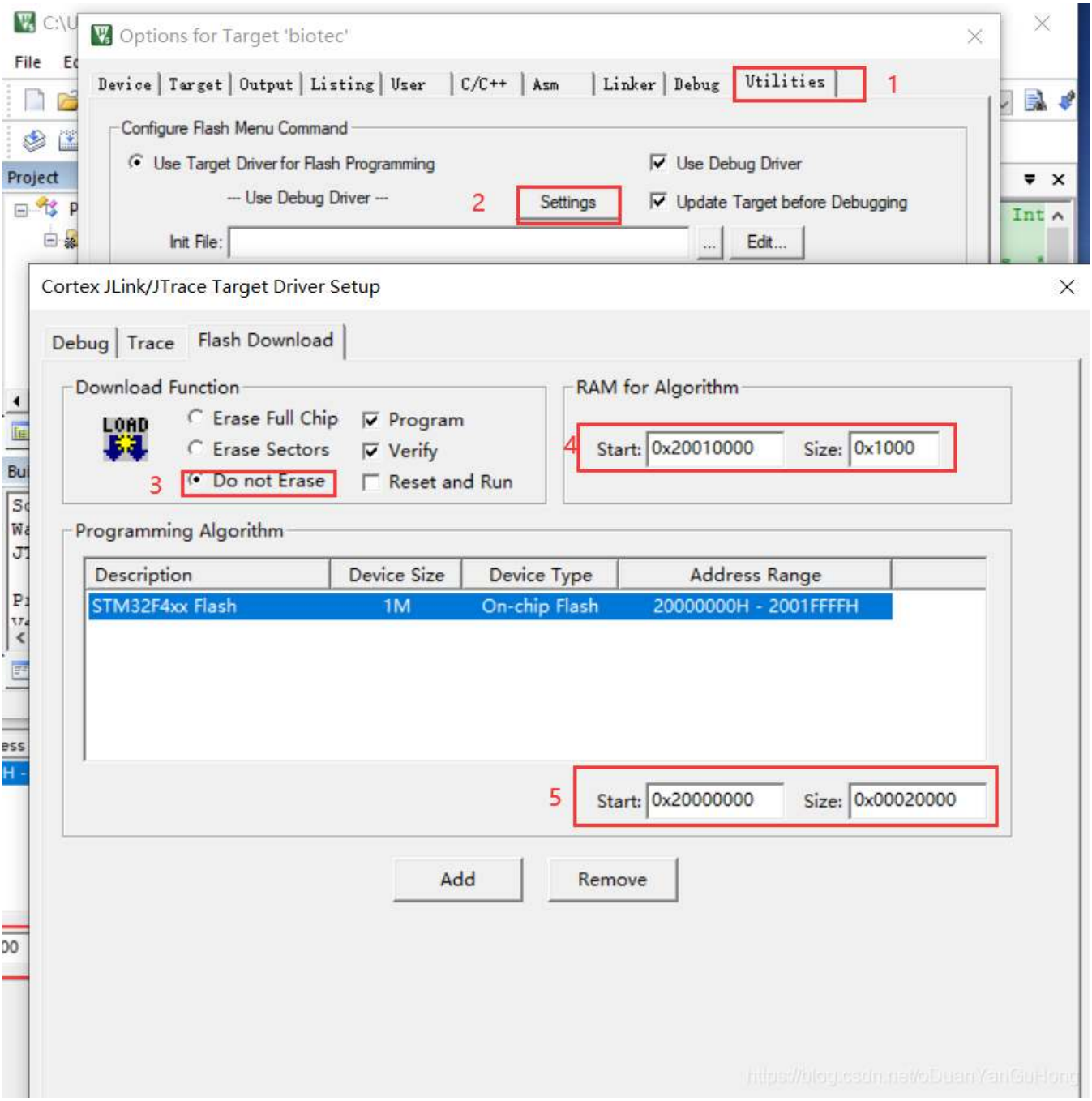


（3） 在 Debug_RAM.ini 文件中輸入如代碼清單 52-4 中的內容。

```
/* Debug_RAM.ini: Initialization File for Debugging from Internal RAM */
/*****************************************************/
/* This file is part of the uVision/ARM development tools. */
/* Copyright (c) 2005-2014 Keil Software. All rights reserved. */
/* This software may only be used under the terms of a valid, current, */
/* end user licence from KEIL for a compatible version of KEIL software */
/*development tools. Nothing else gives you the right to use this software */
/*****************************************************/

FUNC void Setup (void) {
SP = _RDWORD(0x20000000); // 設置棧指針 SP，把 0x20000000 地址中的內容賦值到 SP。
PC = _RDWORD(0x20000004); // 設置程序指針 PC，把 0x20000004 地址中的內容賦值到 PC。
XPSR = 0x01000000; // 設置狀態寄存器指針 xPSR
_WDWORD(0xE000ED08, 0x20000000); // Setup Vector Table Offset Register
```

```
}

LOAD %L INCREMENTAL // 下載 axf 文件到 RAM
Setup(); //調用上面定義的 setup 函數設置運行環境

//g, main //跳轉到 main 函數，本示例調試時不需要從 main 函數執行，注釋掉了，程序從啟動代碼開始執行
```

（4）Utilities->Setting 裡修改調試地址



• 把“Download Function”中的擦除選項配置為“Do not Erase”。這是因為數據寫入到內部 SRAM 中不需要像 FLASH 那樣先擦除後寫入。在本工程中，如果我們不選擇“Do not Erase”的話，會因為擦除過程導致下載出錯。

“RAM for Algorithm”一欄是指“編程算法”(Programming Algorithm)可使用的 RAM 空間，下載程序到 FLASH 時運行的編程算法需要使用 RAM 空間，在默認配置中它的首地址為 0x20000000，即內部 SRAM 的首地址，但由於我們的分散加載文件配置，0x20000000 地址開始的 64KB 實際為虛擬 ROM 空間，實際的 RAM 空間是從地址 0x20010000 開始的，所以這裡把算法 RAM 首地址更改為本工程中實際作為 RAM 使用的地址。若編程算法使用的 RAM 地址與虛擬 ROM 空間地址重合的話，會導致下載出錯。

•“Programming Algorithm”一欄中是設置內部 FLASH 的編程算法，編程算法主要描述了 FLASH 的地址、大小以及扇區等信息，MDK 根據這些信息把程序下載到芯片的 FLASH 中，不同的控制器芯片一般會有不同的編程算法。由於 MDK 沒有內置 SRAM 的編程算法，所以我們直接在原來的基礎上修改它的基地址和空間大小，把它改成虛擬 ROM 的空間信息。

4 處 RAM for Algorithm 是用來在 IRAM1 區域劃分一段空間，用來運行 flash 下載算法（可理解為一個程序），從而給 MCU 下載代碼。但是這個空間只在下載代碼的時候有用，下載完了代碼以後，這段空間就可以被你的 APP 代碼（你下載的代碼）佔用的，相當於釋放了。

確認後，代碼重新編譯就可以了在 SRAM 裡仿真了。

2、STM32 的啟動方式

在前面講解的 STM32 啟動代碼章節瞭解到 CM-4 內核在離開復位狀態後的工作過程如下，見圖 52-1：
(1) 從地址 0x00000000 處取出棧指針 MSP 的初始值，該值就是棧頂的地址。
(2) 從地址 0x00000004 處取出程序指針 PC 的初始值，該值指向復位後應執行的第一條指令。

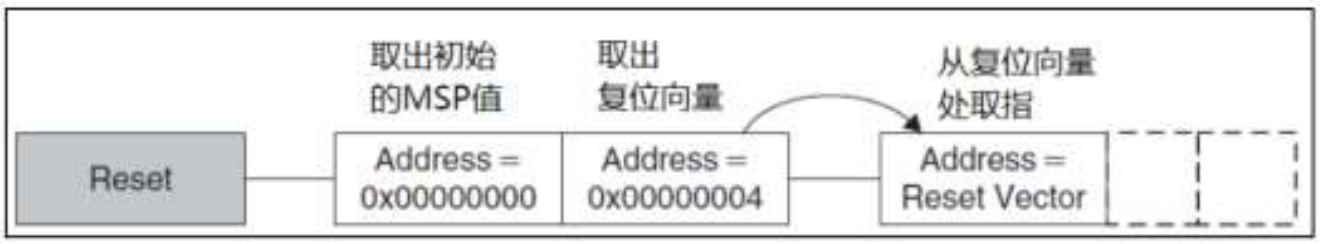


图 52-1 复位序列

上述過程由內核自動設置運行環境並執行主體程序，因此它被稱為自舉過程。
雖然內核是固定訪問 0x00000000 和 0x00000004 地址的，但實際上這兩個地址可以被重映射到其它地址空間。以 STM32F429 為例，根據芯片引出的 BOOT0 及 BOOT1 引腳的電平情況，這兩個地址可以被映射到內部 FLASH、內部 SRAM 以及系統存儲器中，不同的映射配置見表 52-1。

表 52-1 BOOT 引脚的不同设置对 0 地址的映射

BOOT1	BOOT0	映射到的存储器	0x00000000 地址映射到	0x00000004 地址映射到
x	0	内部 FLASH	0x08000000	0x08000004
1	1	内部 SRAM	0x20000000	0x20000004
0	1	系统存储器	0x1FFF0000	0x1FFF0004

内核在離開復位狀態後會從映射的地址中取值給棧指針 MSP 及程序指針 PC，然後執行指令，我們一般以存儲器的類型來區分自舉過程，例如內部 FLASH 啟動方式、內部 SRAM 啟動方式以及系統存儲器啟動方式。

(1) 內部 FLASH 啟動方式

當芯片上電後采樣到 BOOT0 引腳為低電平時， 0x00000000 和 0x00000004 地址被映射到內部 FLASH 的首地址 0x08000000 和 0x08000004。因此，內核離開復位狀態後，讀取內部 FLASH 的 0x08000000 地址空間存儲的內容，賦值給棧指針 MSP，作為棧頂地址，再讀取內部 FLASH 的 0x08000004 地址空間存儲的內容，賦值給程序指針 PC，作為將要執行的第一條指令所在的地址。具備這兩個條件後，內核就可以開始從 PC 指向的地址中讀取指令執行了。

(2) 內部 SRAM 啟動方式

類似地，當芯片上電後采樣到 BOOT0 和 BOOT1 引腳均為高電平時，0x00000000 和 0x00000004 地址被映射到內部 SRAM 的首地址 0x20000000 和 0x20000004，內核從 SRAM 空間獲取內容進行自舉。
在實際應用中，由啟動文件 starttup_stm32f429_439xx.s 決定了 0x00000000 和 0x00000004 地址存儲什麼內容，**鏈接時，由分散加載文件(sct)決定這些內容的絕對地址，即分配到內部 FLASH 還是內部 SRAM**。（下一小節將以實例講解）

(3) 系統存儲器啟動方式

當芯片上電後采樣到 BOOT0 引腳為高電平，BOOT1 為低電平時，內核將從系統存儲器的 0x1FFF0000 及 0x1FFF0004 獲取 MSP 及 PC 值進行自舉。系統存儲器是一段特殊的空間，用戶不能訪問，ST 公司在芯片出廠前就在系統存儲器中固化了一段代碼。因而使用系統存儲器啟動方式時，內核會執行該代碼，該代碼運行時，會為 ISP 提供支持(In System Program)，如檢測 USART1/3、CAN2 及 USB 通訊接口傳輸過來的信息，並根據

這些信息更新自己內部 FLASH 的內容，達到升級產品應用程序的目的，因此這種啟動方式也稱為 ISP 啟動方式。

3、內部 FLASH 的啟動過程

下面我們以最常規的內部 FLASH 啟動方式來分析自舉過程，主要理解 MSP 和 PC 內容是怎樣被存儲到 0x08000000 和 0x08000004 這兩個地址的。

見圖 52-2 ，這是 STM32F4 默認的啟動文件的代碼，啟動文件的開頭定義了一個大小為 0x400 的棧空間，且棧頂的地址使用標號“__initial_sp”來表示；在圖下方定義了一個名為“Reset_Handler”的子程序，它就是我們總是提到的在芯片啟動後第一個執行的代碼。在匯編語法中，程序的名字和標號都包含它所在的地址，因此，我們的目標是把“__initial_sp”和“Reset_Handler”賦值到 0x08000000 和 0x08000004 地址空間存儲，這樣內核自舉的時候就可以獲得棧頂地址以及第一條要執行的指令了。在啟動代碼的中間部分，使用了匯編關鍵字“DCD” 把“__initial_sp”和“Reset_Handler”定義到了最前面的
地址空間。

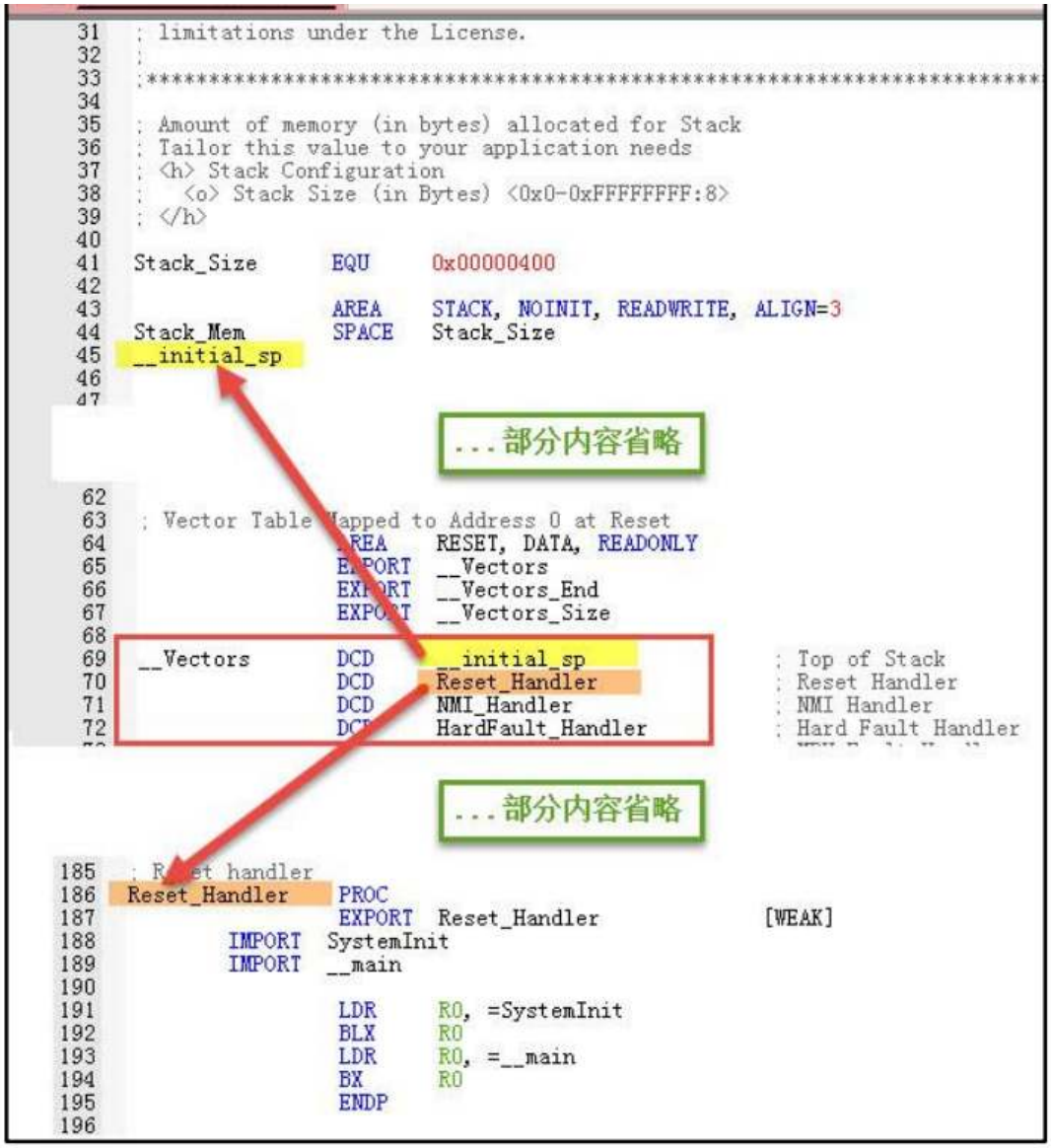


图 52-2 启动代码中存储的 MSP 及 PC 指针内容

在啟動文件中把設置棧頂及首條指令地址到了最前面的地址空間，但這並沒有指定絕對地址，**各種內容的絕對地址是由鏈接器根據分散加載文件(*.sct)分配的**，STM32F429IGT6 型號的默認分散加載文件配置見代碼清單 52-1。

```
//代碼清單 52-1 默認分散加載文件的空間配置
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00100000 { ; load region size_region
    ER_IROM1 0x08000000 0x00100000 { ; load address = execution address
        *.o (RESET, +First)
        *(InRoot$$Sections)
        .ANY (+RO)
    }
```

```
    RW_IRAM1 0x20000000 UNINIT 0x00030000 { ; RW data
        .ANY (+RW +ZI)
    }
}
```

分散加載文件把加載區和執行區的首地址都設置為 0x08000000，正好是內部 FLASH 的首地址，因此匯編文件中定義的棧頂及首條指令地址會被存儲到 0x08000000 和 0x08000004 的地址空間。

類似地，如果我們修改分散加載文件，把加載區和執行區的首地址設置為內部 SRAM 的首地址 0x20000000，那麼棧頂和首條指令地址將會被存儲到 0x20000000 和 0x20000004 的地址空間了。