# [【經驗分享】STM32 keil軟體工具--工程目標選項組態(下）](https://shequ.stmicroelectronics.cn/thread-635371-1-1.html)

[[複製連結]](https://shequ.stmicroelectronics.cn/thread-635371-1-1.html)

[STMCU小助手](https://shequ.stmicroelectronics.cn/space-uid-13573447.html)發佈時間：2022-6-19 21:00

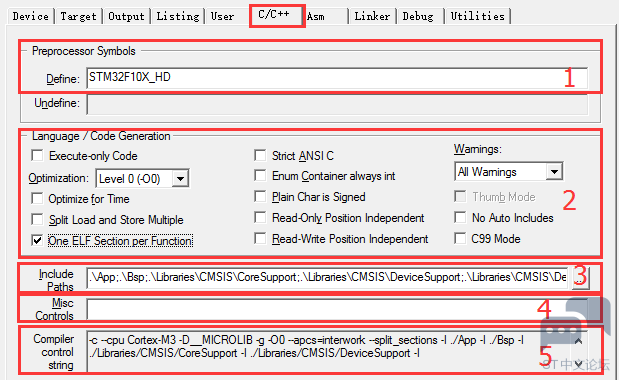
## Ⅰ、寫在前面

本文接著上一篇文章“工程目標選項組態（Ⅰ）”進行講述工程目標選項中後五項的組態，也就是C/C++、Asm、Linker、Debug、Utilities這五項組態的具體內容。

對於工程目標選項中的C/C++選項，不管是初學者，還是資深的軟體工程師都非常重要。初學者要掌握包含路徑、預定義等常規功能。資深軟體工程師對程式碼性能有很高要求，就需要掌握程式碼最佳化等各項組態。

## Ⅱ、C/C++選項

這後面五項中，C/C++選項最為重要，因此部分功能需要重點強調。看選項標題“C/C++”，針對的主要就是C/C++，和後一個選項“Asm”有類似之處。



一張含有 文字 的圖片

自動產生的描述

### 第1處：預處理（Preprocessor Symbols）

這裡主要就是預定義功能，相當於在程序中的#define xxxx。我上面預定義STM32F10X\_HD，在stm32f10x.h檔案中就不用定義了。

### 第2處：語言程式碼生成（Language / Code Generation）

Language/code Generation語言程式碼生成，可以理解成編譯、連結到最後生成程式碼。這部分功能對於程式碼最佳化比較重要，初學者可以不用過多理解，對程式碼大小、運行速度等性能要求較高的人就需要深入理解

* Execute only Code：只生成執行程式碼；

【設定編譯器命令列：--execute\_only】

只生成執行程式碼防止編譯器生成任何資料訪問程式碼部分。

* Optimize：最佳化選擇項，有Level0 - Level3四個選項。

【設定編譯器命令列：-Onum】

初學者、線上偵錯建議使用Level0，也就是不最佳化，這樣執行的效果才和程式碼一樣。如果組態成Level3，線上偵錯可能有些地方最佳化而不能打斷點。

* Optimize for Time：最佳化時間，即最佳化程式碼中費時的地方。

【設定編譯器命令列：-Otime】 --split\_sections

比如有些[演算法](http://lib.csdn.net/base/datastructure" \t "_blank)，本身程式碼量就比較大，運行需要很長時間（假如需要2秒），這個時候勾選上該功能，會發現執行階段間有比較明顯的減少（或許不到1秒時間）。

* Split Load and Store Multiple：載入和儲存多個分裂。

【設定編譯器命令列：--split\_ldm】

非對齊資料採用多次訪問方式。當 LMD/STM 指令有 4 個以上產生時，列分裂LMD 和 STM 指令，以減不中斷延遲。

* One ELF Section per Function：最佳化每一個函數 ELF 段（建議都勾選上）。

【設定編譯器命令列：--split\_sections】

每個函數都會產生一個 ELF 段，勾選上，允許最佳化每一個 ELF 段。這個選項可以減少潛在的共享地址、資料和函數之間的字串。

直白的意思：可以減少程式碼量ROM的大小（記憶體RAM不會減小）。

舉一個例子，勾選之前和勾選之後，編譯後儲存大小對比：

勾選之前：

Program Size: Code=2540 RO-data=336 RW-data=40 ZI-data=1024

勾選之後：

Program Size: Code=908 RO-data=320 RW-data=40 ZI-data=1024

* Strict ANSI C：標準（嚴格）的ANSC。

【設定編譯器命令列：--strict】

也就是說：編譯時嚴格按照標準的ANSI C進行檢查。

* Enum [Container](http://lib.csdn.net/base/docker) always int:列舉總是int型

【設定編譯器命令列：--enum\_is\_int】

很容易理解，我們列舉時成員變數類型為int型。

* Plain Char is Signed：純字元標記為字元。

【設定編譯器命令列：--signed\_chars】

程式碼舉例：char a[] = “abcd”; 也就是說將“abcd”標記為字元型。

* Read-Only Position Independent：為常數生成獨立的程式碼空間。

【設定編譯器命令列：--apcs=/ropi】

比如：我們定義字庫變數為常數，勾選該選項，會將這些字庫變數放在獨立的程式碼空間。

* Read-Write Position Independent：為可讀寫程式碼生成獨立的程式碼空間。

【設定編譯器命令列：--apcs=/rwpi】

* Warnings：警告

【No Warnings設定編譯器命令列：-W】

No Warnings：不會有警告提示和輸出；

All Warnings：所有警告提示和輸出。

* Thumb Mode：Thumb模式。

指定設定檔案或資料夾（組）為Thumb模式。

【注意：在工程中該模式為默認，也就是不能選擇】

* No Auto Includes：不自動新增標頭檔（一般不勾選）。

不勾選該選項，編譯器就會在Keil安裝路徑尋找你工程中.h檔案。

舉例：我們定義uint8\_t是定義在stdint.h檔案裡面的，但是我們工程目錄下一般是沒有stdint.h檔案。這時候，編譯器就會在Keil路徑下去尋找stdint.h檔案。

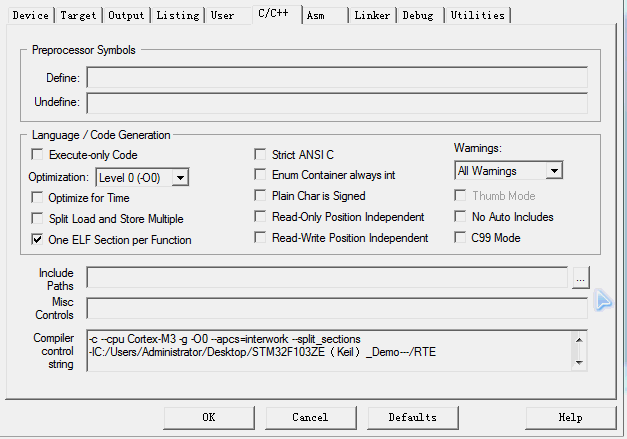
* C99 Mode:C99標準模式。

【設定編譯器命令列：--c99】

C語音有標準有多個版本，如C89、C90、C99等。

### 第3處：包含路徑（Include Paths）

包含路徑是使用Keil（及類似）軟體必須掌握的一項。包含路徑就是指定我們工程中使用檔案所在的位置，讓編譯器找到相應的檔案。是初學者、高級軟體工程師都必須掌握的一項。



一張含有 文字 的圖片

自動產生的描述

### 第4處：多功能控制項（Misc Controls）

指定沒有單獨的對話方塊控制項。例如：錯誤消息用日本語言來顯示消息。【不常用】

### 第5處：編譯器控制字串（Compiler control string）

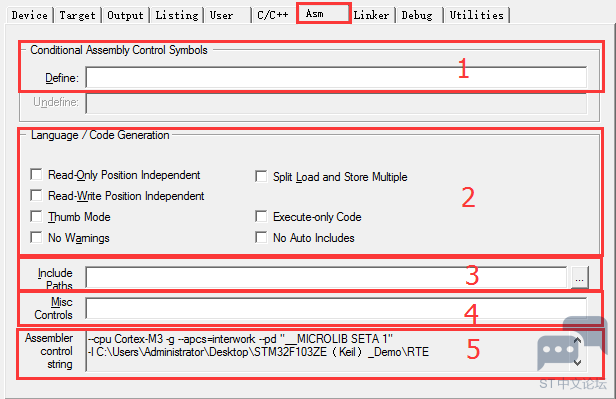
這裡是針對編譯器執行的命名，顯示當前在編譯器命令列指令。

在上面“第2處：語言程式碼生成”中有一個中括號【設定編譯器命令列：】

裡面的命名就顯示在這裡。

## Ⅲ、Asm選項

從這個選項的排版和結構可以知道，和前面一個選項“C/C++”類似，只是這裡針對的是Asm。因此，這一章節只簡述，不會重點強調，可以參看上一章節進行對比，第3、4、5處和C/C++選項一樣，這裡不講述。



一張含有 文字 的圖片

自動產生的描述

### 第1處：有條件的裝配控制符號（Conditional Assembly Control Symbols）

指定彙編條件，這裡類似上一章節C/C++選項中的預處理。

### 第2處：語言程式碼生成（Language / Code Generation）

Read-Only Position Independent：為常數生成獨立的程式碼空間。

Read-Write Position Independent：為可讀寫程式碼生成獨立的程式碼空間。

Thumb Mode：Thumb模式。

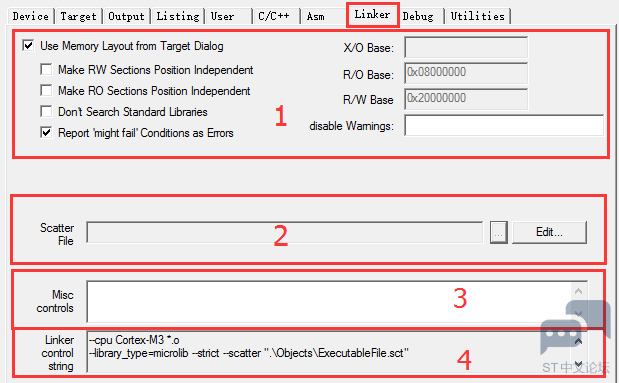
Split Load and Store Multiple：載入和儲存多個分裂。

Execute only Code：只生成執行程式碼；

No Auto Includes：不自動新增標頭檔（一般不勾選）。

## Ⅳ、Linker選項

這個選項Linker連結，也就是是連結器組態選項。可以修改、編輯和查看連結的檔案。第1、2處是重點，第3、4處和C/C++選項一樣，這裡不講述。



一張含有 文字 的圖片

自動產生的描述

### 第1處：使用分散檔案載入對話方塊Target頁面（Use Memory Layout from Target Dialog）

* Make RW Sections Position Independent：使RW段獨立。

【設定編譯器命令列：--rwpi】

啟用時：變數區域（包含RW和ZI）具有獨立地址。

停用時：變數區域（包含RW和ZI）位於絕對的記憶體地址。

* Make RO Sections Position Independent：使RO段獨立。

【設定編譯器命令列：--ropi】

啟用時：常數和程式碼區域（RO）具有獨立地址。

停用時：常數和程式碼區域（RO）位於絕對的記憶體地址。

* Don't Search Standard Libraries：不搜尋標準庫。

【設定編譯器命令列：--noscanlib】

停用默認編譯器執行階段庫的掃描。

* Report 'might fail' Conditions as Errors：報告'might fail'條件認為是錯誤。

【設定編譯器命令列：--strict】

報告的條件可能導致失敗的錯誤，而不是警告。

* X/O Base：X/O基地址。

【設定編譯器命令列：--xo\_base=address】

* R/O Base：R/O基地址。

【設定編譯器命令列：--ro\_base=address】

* R/W Base：R/W基地址。

【設定編譯器命令列：--rw\_base=address】

* disable Warnings:警用警告。

【設定編譯器命令列：--diag\_suppress】

### 第2處：分散檔案（Scatter File）

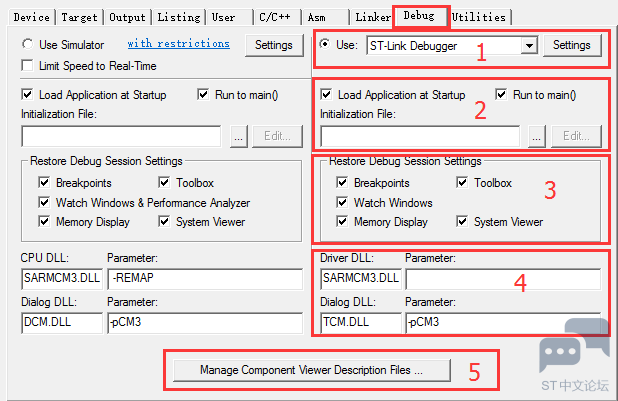
這裡可以載入、查看和編輯分散檔案。點選後面就的三點“...”可以載入檔案；點選“Edit...”查看和編輯對應的檔案。





## Ⅴ、Debug選項

這個選項比較重要，主要用於（軟體模擬、硬體線上）偵錯使用。由於軟體模擬和硬體線上偵錯組態介面基本一致，而現在我們基本都是硬體線上偵錯。因此，只選擇硬體線上偵錯介面進行講述。

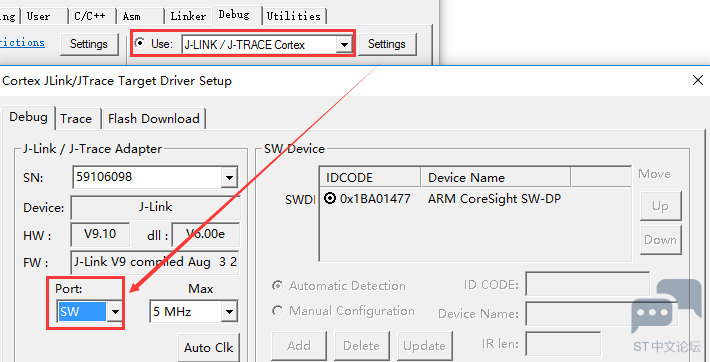


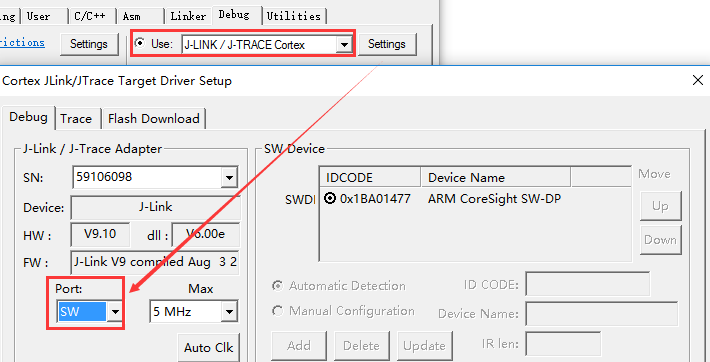
一張含有 文字 的圖片

自動產生的描述

### 第1處：選擇硬體線上偵錯

下載偵錯程式的選擇不用多說，主要說一下後面“Setting”。很多人常用J-Link下載偵錯程式，而偵錯STM32時，可以使用四線SWD模式。如果使用J-Link進行SWD偵錯。這個時候就需要在“Setting”裡面選擇“SW”模式，如下如：





### 第2處：選擇硬體線上偵錯

Load Application at Startup：啟動時載入應用程式。

Run to main()：程序執行到main()函數。

進入偵錯模式時，程序自動運行到main函數處。

Initialization File：載入、編輯初始化檔案。

這裡在某些情況下可以使用，比如：在RAM中偵錯程式碼。

### 第3處：復位偵錯會話設定（Restore Debug Session Settings）

這裡復位設定就是恢復設定的意思，如果勾選上，點選一下“復位”就會恢復到之前的狀態。包括：斷點Breakpoints、窗口Watch Windows、性能分析器 Performance Analyzer、記憶體窗口Memory Window、工具箱Toolbox、系統查閱器System Viewer等。

### 第4處：DLL檔案（最好默認）

這裡的組態屬於Keil自身的組態，最好不要修改。

CPU/Driver DLL - Parameter：CPU驅動檔案和參數。

Dialog DLL - Parameter：會話框DLL檔案和參數。

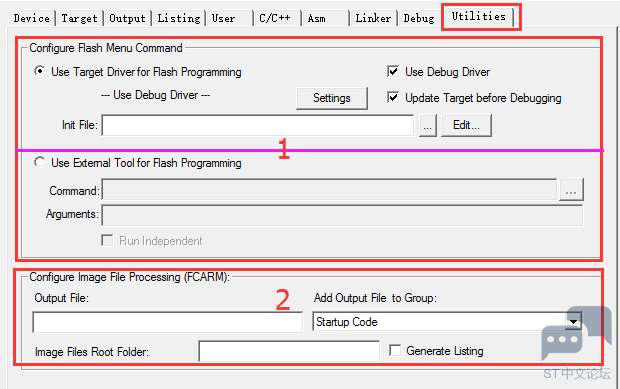
### 第5處：管理元件描述檔案

Manage Component Viewer Description Files

這裡一般不用去管理。

## Ⅵ、Utilities選項

本章節內容公共選項，內容比較簡單。



一張含有 文字 的圖片

自動產生的描述

### 第1處：組態FLASH菜單命名（Configure Flash Menu Command）

這裡是二選一選項，一般我們使用上面的“Update Target Driver for Flash Programming”。

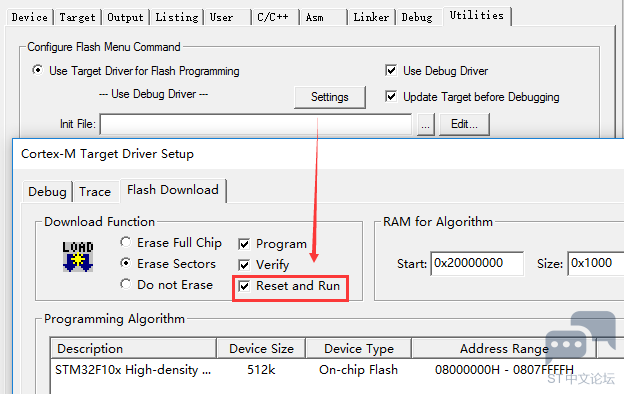
Use Debug Driver：使用偵錯驅動。

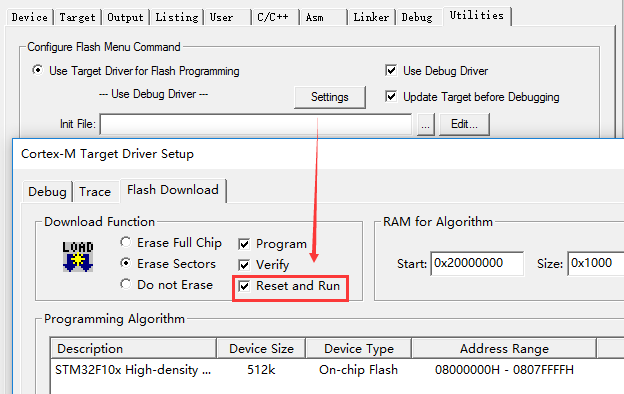
Update Target Before Debugging：偵錯之前更新目標。

一般都勾選上，因為我們下載程序之前檢測到程式碼修改了，就會重新編譯程序（也就是更新目標）

Setting：設定。

很多人下載程序之後，需要復位一下程序才運行，原因在於沒有勾選“Reset and Run”，如下圖：





### 第2處：組態圖像檔案的處理（Configure Image File Processing）

這個選項我們不常用，感興趣的朋友可以自行瞭解一下。