关于 STM32CubeIDE 链接脚本的小问题

关键字：Linker Script，NOLOAD

1. 概述

越来越多的客户在使用 STM32CubeIDE 作为集成开发工具。STM32CubeIDE 在编译代码的时候，用到了链接脚本。通常情况下，STM32CubeIDE 会自动生成默认的链接脚本。但是有些情况下，例如，用户程序需要定义一些特别的段来放置代码或者数据的时候，我们就需要修改链接脚本文件。

最近有客户在修改链接脚本后，编译没有出现问题。但是编译之后生成的 BIN 文件很大，导致无法烧录到 Flash 中。结合这个问题，本文详细分析一下它的原因以及解决办法。

2. 问题描述

使用 STM32CubeIDE 创建工程的时候，在项目工程目录文件夹下生成后缀为 ld 的链接脚本文件，程序的编译和链接都会依赖链接脚本文件。如下图所示，STM32H743VIT6 的 RAM 空间被包含 6 块，每块 RAM 的起始地址是独立的，如果客户需要把指定特定的 RAM 区域放置数据或者代码的时候，需要手动修改链接脚本文件。

图1. Memory Mapping for STM32H43

	Memory	Size in Kbytes	Start address
RAM area	Backup SRAM	4	0x3880 0000
	SRAM4	64	0x3800 0000
	SRAM2	16	0x3002 0000
	SRAM1	32	0x3000 0000
	AXI-SRAM	384	0x2400 0000
	DTCM	128	0x2000 0000
Code area	System memory 2		0x1FF4 0000
	System memory		0x1FF0 0000
	Flash memory	2048	0x0800 0000
	ITCM	64	0x0000 0000

客户在使用 STM32H743VIT6 进行应用开发的时候，需要在 0x24000000 以及 0x30000000 的 RAM 空间定义两个未初始化的数组。所以客户修改了 ld 链接脚本文件，添加两

个 SECTIONS 分别定位在 0x24000000 和 0x30000000 位置，同时在代码中使用__attribute__ 关键字指定数组在对应的 SECTIONS 中。

依照客户的问题描述，我们首先修改链接脚本，添加两个 SECTIONS 分别为 ram_at_0x24000000 和 ram_at_0x30000000，参考如下。

图2. ld 链接脚本文件

```
165   .ram_at_0x24000000 :
166   {
167     *(.user_test_data)
168   } >RAM_D1
169
170   .ram_at_0x30000000 :
171   {
172     *(.user_test_data2)
173   } >RAM_D2
174
```

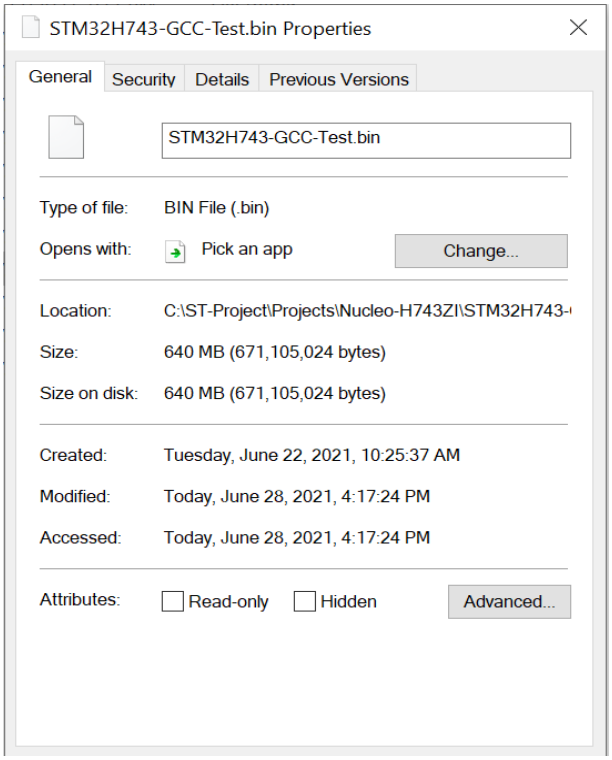
同时在源代码中添加两个数组，分别定位在添加的 SECTIONS 中，参考如下。

图3. 定义数组参考代码

```
102 int RamDataA[4096] __attribute__((section(".ram_at_0x24000000")));
103 int RamDataB[4096] __attribute__((section(".ram_at_0x30000000")));
```

编译修改之后的工程，是可以正常运行的，但是我们发现当把 ELF 目标文件转为 BIN 文件之后，产生的 BIN 文件非常的大。这就导致如果使用 BIN 文件进行下载的话，是无法下载成功的，因为 BIN 文件的大小以及超出了 Flash 的存储空间。由下图 BIN 文件属性我们可以看到 BIN 文件的大小为 650M。

图4. BIN 文件大小



那么是什么原因导致产生的 BIN 文件这么大呢？文件就出在 ld 链接脚本文件这里。

3. 问题分析

BIN 文件是最纯粹的二进制机器代码, 或者说是"顺序格式"。按照 assembly code 顺序翻译成 binary machine code，内部没有地址标记。Bin 是直接的内存映象表示，二进制文件大小即为文件所包含的数据的实际大小。 BIN 文件就是直接的二进制文件，一般用编程器烧写时从 00 开始，而如果下载运行，则下载到编译时的地址即可。

STM32CubeIDE 依赖链接脚本文件生成目标 ELF 文件，生成 ELF 目标文件之后，STM32CubeIDE 会依赖 GNU Objcopy 工具把 ELF 文件转为 BIN 文件。这个过程简单一点来说，就是提取 ELF 文件中的代码段以及可加载数据段按照地址信息顺序生成 BIN 文件。

所以，如果 BIN 文件很大，那说明可加载的代码段和数据段很大。在我们的测试项目中，代码段是固定的；现在 BIN 文件很大，说明是数据段过大导致的。

通过查看编译产生的 list 文件，我们可以清楚的看到每个段的大小。

图5. Sections Map

```
STM32H743-GCC-Test.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .isr_vector   00000298  08000000  08000000  00010000  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .text         00005bbc  08000298  08000298  00010298  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .rodata       00000038  08005e54  08005e54  00015e54  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .ARM          00000008  08005e8c  08005e8c  00015e8c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .init_array   00000004  08005e94  08005e94  00015e94  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .fini_array   00000004  08005e98  08005e98  00015e98  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .data         00000010  20000000  08005e9c  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .RxDecripSection 00000060  20000010  08005eac  00020010  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .TxDecripSection 00000060  20000070  08005f0c  00020070  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .bss          00000578  200000d0  08005f6c  000200d0  2**2
                  ALLOC
 10 ._user_heap_stack 00000600  20000648  08005f6c  00020648  2**0
                  ALLOC
 11 .ARM.attributes 0000002e  00000000  00000000  00044000  2**0
                  CONTENTS, READONLY
 12 .ram_at_0x24000000 00004000  24000000  24000000  00030000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 13 .ram_at_0x30000000 00004000  30000000  30000000  00040000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
 14 .debug_info   0001f101  00000000  00000000  0004402e  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 15 .debug_abbrev 0000301b  00000000  00000000  0006312f  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 16 .debug_aranges 00000ef8  00000000  00000000  00066150  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 17 .debug_ranges 00000e20  00000000  00000000  00067048  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 18 .debug_macro  00036a79  00000000  00000000  00067e68  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 19 .debug_line   00011a87  00000000  00000000  0009e8e1  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 20 .debug_str    0015c3e6  00000000  00000000  000b0368  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 21 .comment      00000053  00000000  00000000  0020c74e  2**0
                  CONTENTS, READONLY
 22 .debug_frame  00003f44  00000000  00000000  0020c7a4  2**2
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

从上图的 Sections Map 中，每个 Section 会有几个关键的参数和类型。Size 为每个 Section 的大小，VMA 是指 Section 在程序运行时候的地址，LMA 是指 Section 在 Flash 中的加载地址。通常情况下从 Flash 执行的程序，Code 段 VMA 和 LMA 是一样的。Data 段的 VMA 会放在 RAM 中，LMA 会放在 Flash 中，所以 Data 段的 VMA 和 LMA 通常不一样。在每个段的类型中，还有一个关键的属性是 LOAD，如果定义了 LOAD 属性，那说明这个段是需要写入 Flash 中的。

GNU Objcopy 工具生成 BIN 文件的过程，实际上就是把 ELF 文件中各个定义了 LOAD 属性的 SECTION 按照 LMA 的先后顺序提取出来，写入 BIN 文件中。SECTION 的地址如果不是连续的，则会填充 DUMMY 数据。所以 BIN 文件的大小为最大 LMA 加上对应的 SECTION 的 Size。

结合上图，该工程最终生成的 BIN 文件大小为 0x30000000+0x4000 -0x8000000 = 671105024 和图 4 的文件属性显示的大小是一致的。

4. 问题解决

根据上面章节分析的原因，BIN 文件过大是新增加的两个 SECTIONS 导致的。如图 5 所示，ram_at_0x24000000 段和 ram_at_0x30000000 段都具有 LOAD 属性，所以 GNU Objcopy 工具认为该段是需要加载到 Flash 中的。如果修改该段的属性为 NOLOAD，则可解决这个问题。

参考 The GNU Linker 文档第 73 页，指定 SECTION 的类型为 NOLOAD。最终修改后的链接脚本文件如下所示。

图6. 更新后的链接脚本

```
165   .ram_at_0x24000000 (NOLOAD) :
166   {
167     *(.user_test_data)
168   } >RAM_D1
169
170   .ram_at_0x30000000 (NOLOAD) :
171   {
172     *(.user_test_data2)
173   } >RAM_D2
```

依赖修改后的链接脚本编译得到的目标 ELF 文件，查看其段描述，可以知道新添加的段的属性不再为 LOAD。GNU Objcopy 工具在进行 BIN 文件生成的时候，也不会加载这两个段。所以，最终得到的 BIN 文件只包含有效的代码段和数据段，大小为 24428 字节。

图7. 更新后的段描述

```
STM32H743-GCC-Test.elf:     file format elf32-littlearm

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .isr_vector   00000298  08000000  08000000  00010000  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .text         00005bbc  08000298  08000298  00010298  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .rodata       00000038  08005e54  08005e54  00015e54  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .ARM          00000008  08005e8c  08005e8c  00015e8c  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .init_array   00000004  08005e94  08005e94  00015e94  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .fini_array   00000004  08005e98  08005e98  00015e98  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .data         00000010  20000000  08005e9c  00020000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .RxDecripSection 00000060  20000010  08005eac  00020010  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  8 .TxDecripSection 00000060  20000070  08005f0c  00020070  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  9 .bss          00000578  200000d0  08005f6c  000200d0  2**2
                  ALLOC
 10 ._user_heap_stack 00000600  20000648  08005f6c  00020648  2**0
                  ALLOC
 11 .ARM.attributes 0000002e  00000000  00000000  000200d0  2**0
                  CONTENTS, READONLY
 12 .ram_at_0x24000000 00004000  24000000  24000000  00030000  2**2
                  ALLOC
 13 .ram_at_0x30000000 00004000  30000000  30000000  00030000  2**2
                  ALLOC
 14 .debug_info   0001f101  00000000  00000000  000200fe  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 15 .debug_abbrev 0000301b  00000000  00000000  0003f1ff  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 16 .debug_aranges 00000ef8  00000000  00000000  00042220  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 17 .debug_ranges 00000e20  00000000  00000000  00043118  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 18 .debug_macro  00036a79  00000000  00000000  00043f38  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 19 .debug_line   00011a87  00000000  00000000  0007a9b1  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 20 .debug_str    0015c3e6  00000000  00000000  0008c438  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 21 .comment      00000053  00000000  00000000  001e881e  2**0
                  CONTENTS, READONLY
 22 .debug_frame  00003f44  00000000  00000000  001e8874  2**2
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

5. 小结

本文通过一个具体的问题阐述了 BIN 文件的产生过程，同时对链接脚本的格式做了一个简单的介绍。STM32 用户后期如果遇到变量无需加载却导致 BIN 文件过大的问题，可依照该方法进行处理。

参考文献

文件编号	文件标题	版本号	发布日期
	The GNU linker	2.30.0	

文档中所用到的工具及版本

STM32CubeIDE 1.6.1

版本历史

日期	版本	变更
2021 年 10 月 29 日	1.0	首版发布