AN0004
应用笔记

AT32 Performance Optimization

前言

这篇应用笔记描述了如何通过软件方法提高AT32的运行效能。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32F403A 系列 AT32F407 系列 AT32F413 系列 AT32F435 系列 AT32F437 系列 AT32A403A 系列

目录

表目录

图目录

1 AT32 性能优化概述

性能提升是多方面调优共同作用的结果。在着手优化之前需要对整个系统的软硬件结构和参数有深入的了解。硬件需要对如Flash大小、SRAM大小、零等待区和非零等待区大小、主频等参数等有准确的认识，软件需要对整个流程熟悉，对代码、算法的执行时间和重要数据的访问频次等信息有个大致的判断。第二步再结合工程内容的实际情况具体分析，一步步进行系统优化，以达到提升性能的目的。实际优化过程可分为如下几个大的步骤：

1） 在未优化情况下生成bin文件，比较大小判断是否需要更进一步优化。

2） 是否进行SRAM扩展。

3） 查看map映射表，大致了解链接结构。

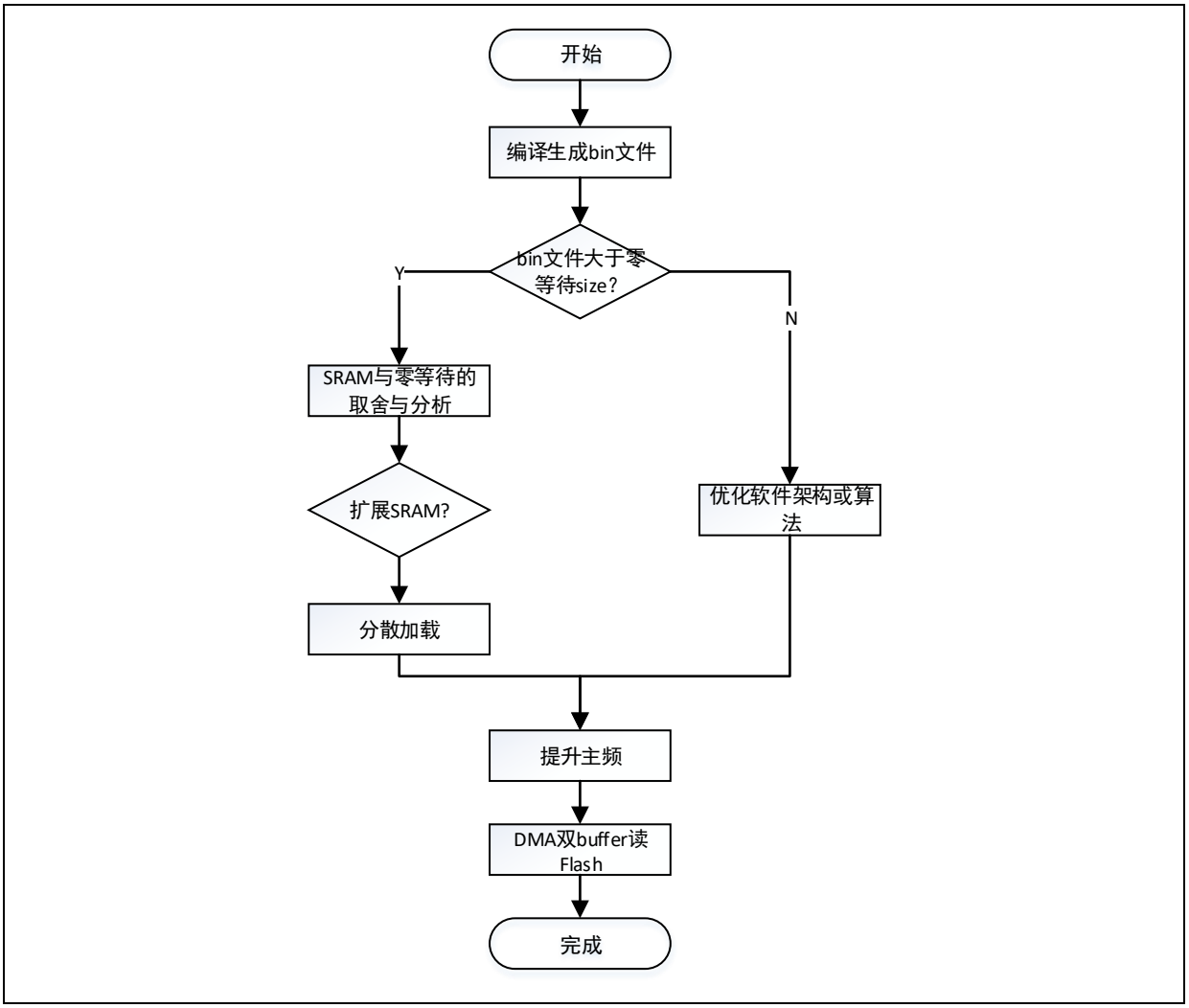
4） 调整代码链接结构，分散加载。

5） 提高系统运行频率。

6） 大量Flash数据读取时，采用DMA双Buffer方式。

大致的优化步骤及流程如下图：

图 1. 优化流程图



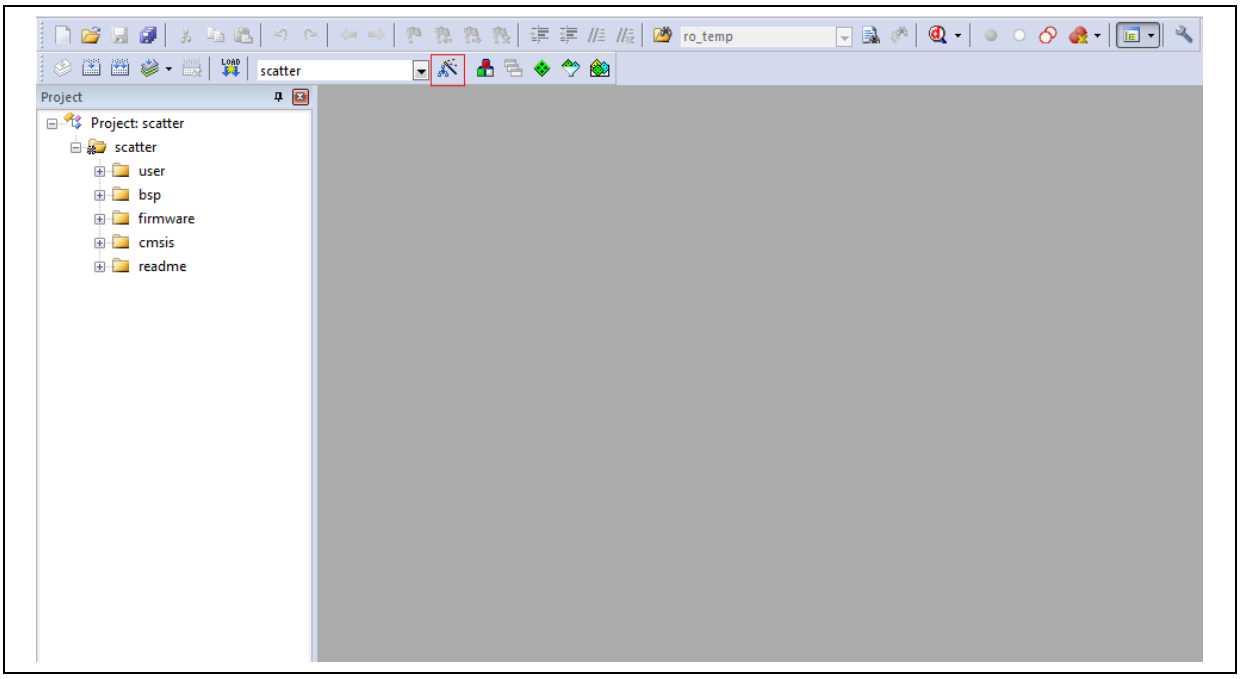
2 Bin 文件判断

2.1 生成 bin 文件

使用keil开发环境在build编译后并生成芯片烧录的二进制文件。

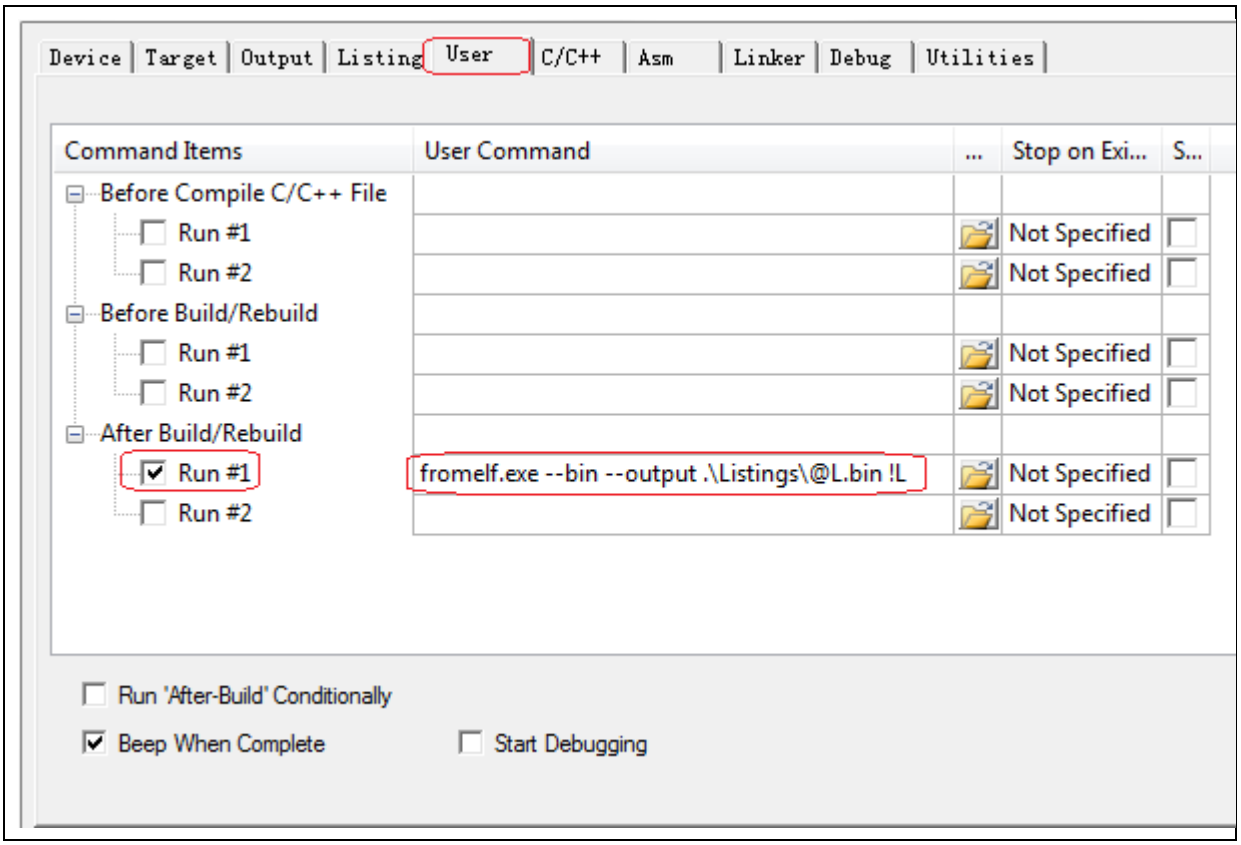
用keil打开工程后，点击Options for Target图标，如下图所示

图 2. OptionForTarget 图标



点击 Options for Target 后会出现如下配置窗口，选择 User 选项卡，并勾选 After Build/Rebuild 一栏下的 Run #1，且在后面的文本框内输入以下命令行：fromelf.exe --bin --output .\Listings\@L.bin !L。操作如下图所示：

图 3. 编译命令



2.2 分析 bin 大小

在编译之后找到生成的二进制文件，并查看生成的bin文件大小。因AT32F部分系列MCU（按具体型号datasheet为准）内部实现了零等待存储空间，所以当bin文件大小在小于零等待存储空间大小的情况下无需更进一步的对执行文件的链接结构进行优化调整，这种情况下更多的应该考虑软件架构和算法等的优化，或者可以尝试提升主频的办法来达到更好的运行性能。

后面的内容我们都将围绕在bin文件大小在大于零等待存储空间的情况下来对执行文件的结构优化、SRAM需求与取舍进行讨论。

3 SRAM 扩展

对于AT32F部分系列MCU，在默认情况下片内包含SRAM和零等待存储空间（Size大小以具体型号为准，请查看datasheet），在特殊需求下用户可以自行的选择对SRAM和零等待存储空间的大小进行重新配置，空间示例如下：

图 4. AT32F403A SRAM 和零等待配置

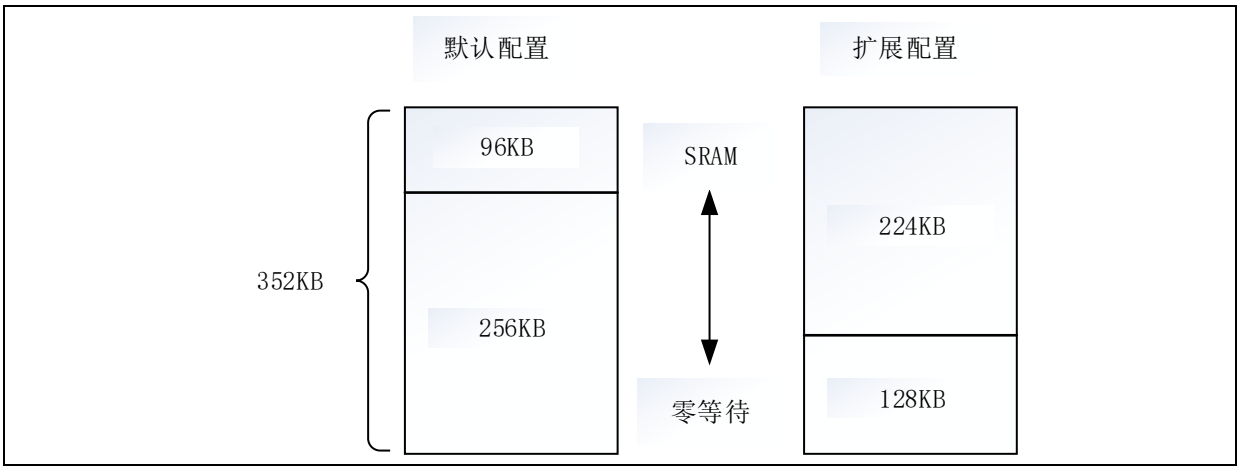


图 5. AT32F413 SRAM 和零等待配置

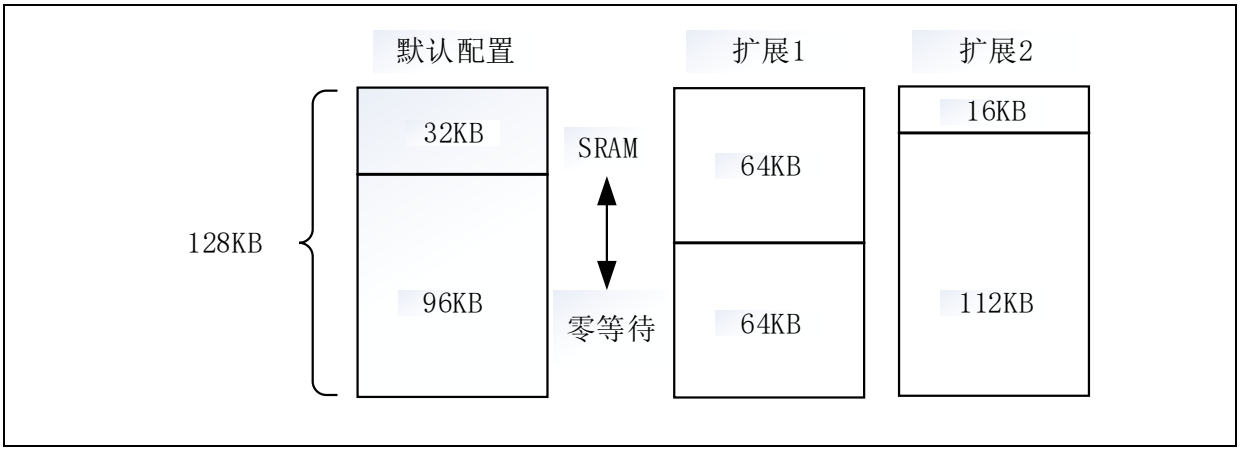
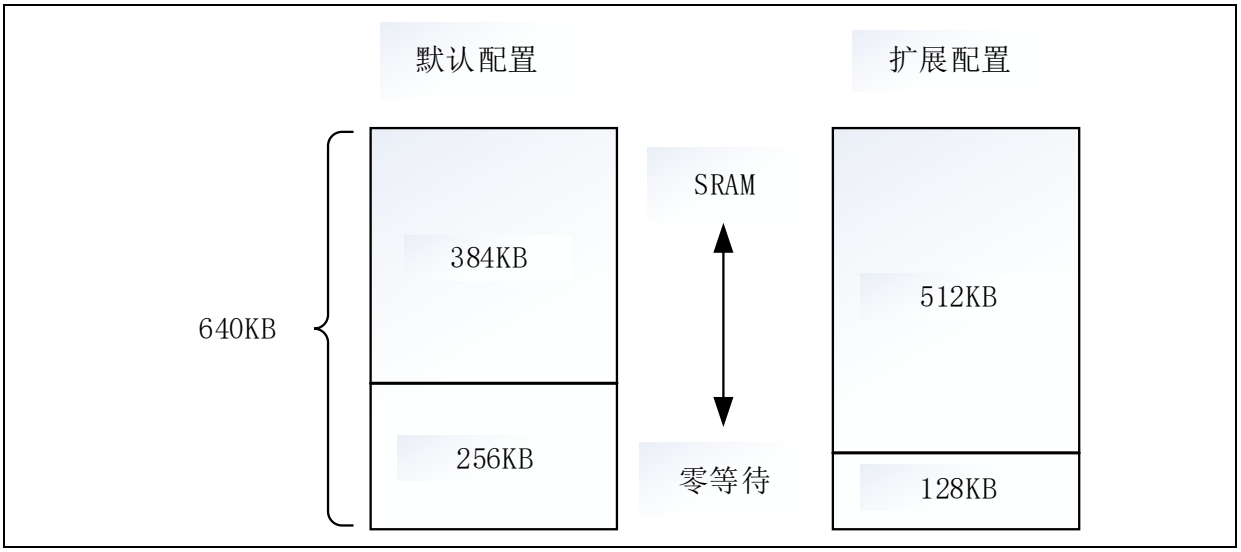


图 6. AT32F435 SRAM 和零等待配置



3.1 SRAM 需求分析与取舍

零等待区是芯片内部以实现快速启动和高速运行而在Flash与SRAM之间做的一个预加载存储区。在芯片上电后，硬件自动从Flash的起始地址拷贝零等待区大小的数据到零等待存储区里面，以备系统后期运行时能对预加载数据的快速访问。

SRAM是系统运行时的存储区域，其中存储的数据包括指令、数据、堆栈等信息。

在此看来对于两片存储区的大小都是越大越好，为了做出合理的分配和取舍，建议在以下几种情况下进行SRAM扩展配置：

1) 当bin文件远大于默认零等待区间大小的情况。
2) 当高频率使用数据大于或接近于默认零等待区间大小的情况。
3) 当系统运行存在操作系统且具有大量的进程调度和临时变量的情况。
4) 当系统运行存在较多函数嵌套和中断且有大量临时变量的情况。

3.2 扩展配置方法

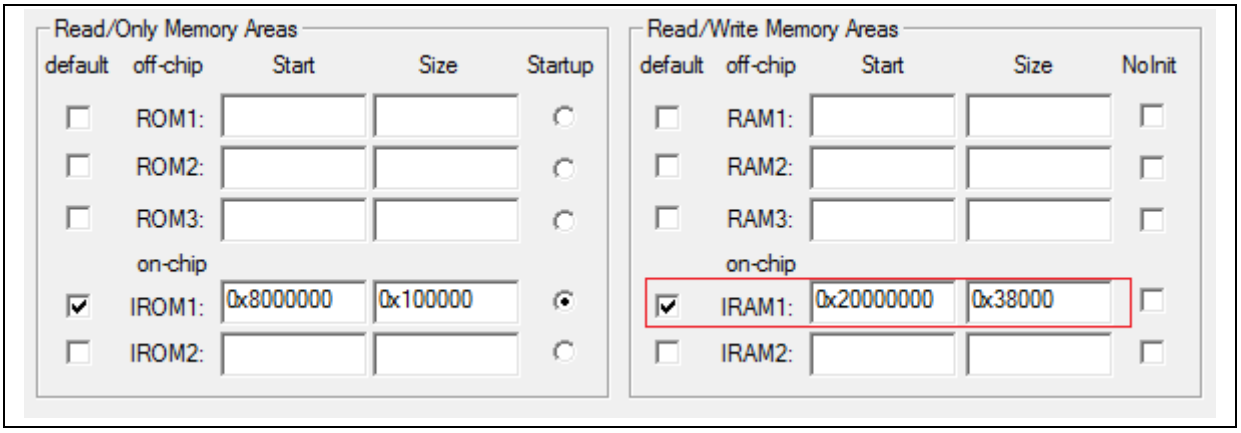
SRAM扩展方法请参考《AN0026_Extending_SRAM_in_User's_Program》。

3.3 SRAM 扩展后注意事项

使用Keil集成开发环境作为开发工具时，在选定了Pack文件时即选定了默认的SRAM编译链接范围，然而由于硬件SRAM资源是可调整的，Pack中只能默认一种设置，故实际情况与默认设置可能存在差异。针对此情况，以下采用AT32F403A系列MCU进行示例说明，其余系列注意事项与此类似：

在工程中点击Options for Target后，选择Target选项卡，可以看到如下图所示：

图 7. Keil 下 SRAM 配置框



红色方框内展示了AT32 Pack文件指定的默认SRAM的起始地址和大小，由此可以看出软件代码在编译链接时默认使用的SRAM大小为0x38000（224KB），然而从前面的介绍中可以知道，AT32F403A系列MCU默认SRAM大小为0x18000（96KB），可扩展为224KB。故使用AT32的Pack文件时有以下两种情况：

a) 使用的SRAM为扩展模式（224KB），软件编译后链接的运行地址和物理地址范围一致，此情况下不会出现任何问题。

b) 使用SRAM为默认模式（96KB），只要软件代码中的RAM需求小于96KB时不会出现问题，当SRAM需求大于96KB，且不超过224KB，编译链接的过程也是不会出现任何错误提示和警告，但由于链接的运行地址和实际物理地址不对应，此bin文件下载到MCU运行就会发生错误，所以使用过程中需要注意到这种无提示的隐藏问题。为避免这种问题的产生，要么修改上图红色框内

的Size栏处为0x18000，要么将SRAM扩展为224KB。目的是保证实际物理SRAM大小和Keil配置的Size大小一致。

4 分散加载

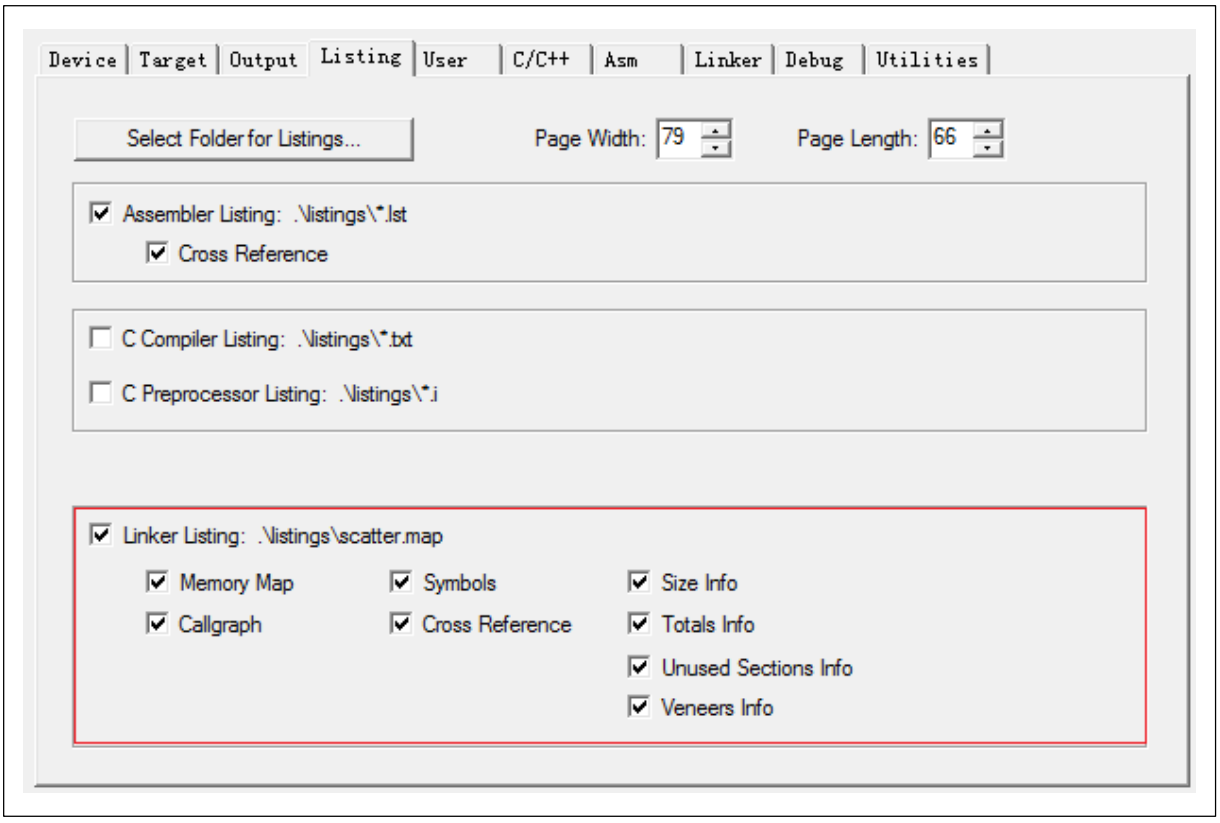
代码分散加载可以以个人主观意识对代码执行频率及效率的理解，人为的调整执行文件的链接结构，以达到提升执行效率及性能的目的。要求对存储空间的地址分布及大小有足够的了解。

4.1 查看 map 映射关系

学习查看map映射关系就为了能更好的理解可执行文件的组成及分布结构，以为后续的调优和分散加载做准备。

在Keil中，完成对源代码的编译后，链接器在链接各目标文件的同时会生成一份map文件，它主要包含交叉链接信息，查看该文件可以了解工程中各种符号之间的引用以及整个工程的Code、RO Data、RW Data以及ZI Data的详细及汇总信息。它的主要内容包含了“节区交叉引用”、“删除无用节区”、“符号映像表”、“存储分布表”、“映像组件大小”等信息。Keil的工程配置默认会生成map信息，具体的配置方式如下：

图 8. Keil 编译信息输出项配置



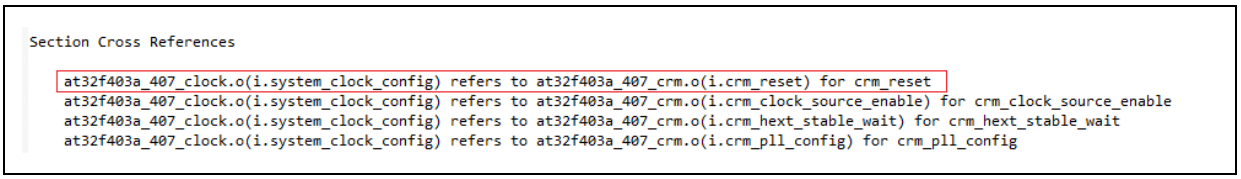
选择Options for Target后点击Listing选项卡，可以自主的选择在map文件中输出以上的信息。

在编译生成map文件后，可以选择用记事本打开查看map信息。

1) 节区交叉引用

在这部分中，详细的列出了各个*.o文件之间的符号引用，由于*.o文件是由.s或c/c++源文件编译生成的，各个文件及文件内的节区间互相独立，链接器根据它们之间的互相引用链接起来，链接的详细信息就记录在这里。如下图说明：

图 9. 符号引用

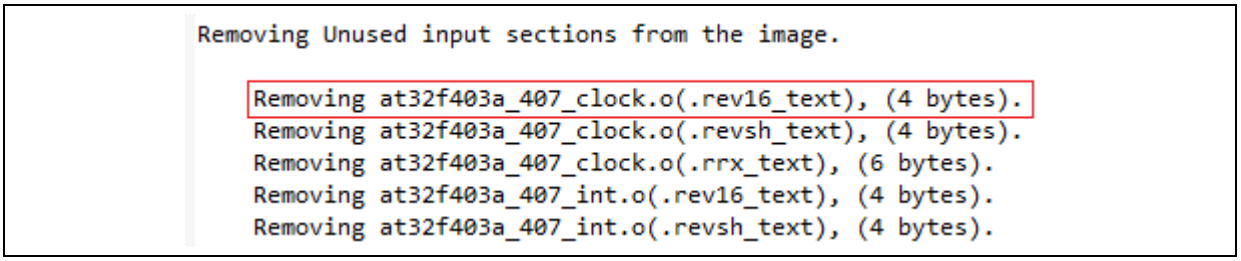


红色标记部分的意思是at32f403a_407_clock.c文件中的system_clock_config函数调用了at32f403a_407_crm.c文件中的crm_reset函数。

2) 删除无用节区

这部分列出了在链接过程中发现工程中未被引用的节区，这些未被引用的节区将会被删除。其主要意思指将未引用的节区不加入到*.axf文件中，而不是从*.o文件中将其删除。这样可防止这些无用的数据占用程序空间。如下图说明：

图 10. 删除无用节区

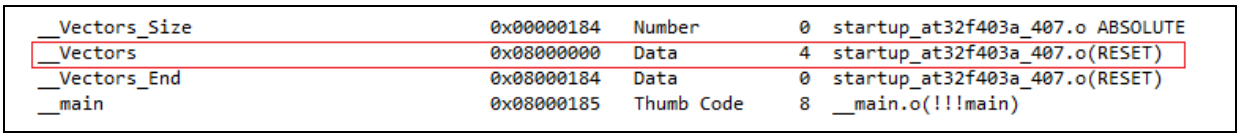


这里表面在at32f403a_407.o目标文件里有4字节的未引用节区，不会将这部分链接到最终的*.axf文件。

3) 符号映像表

符号映像表截取如下图所示：

图 11. 符号映像表

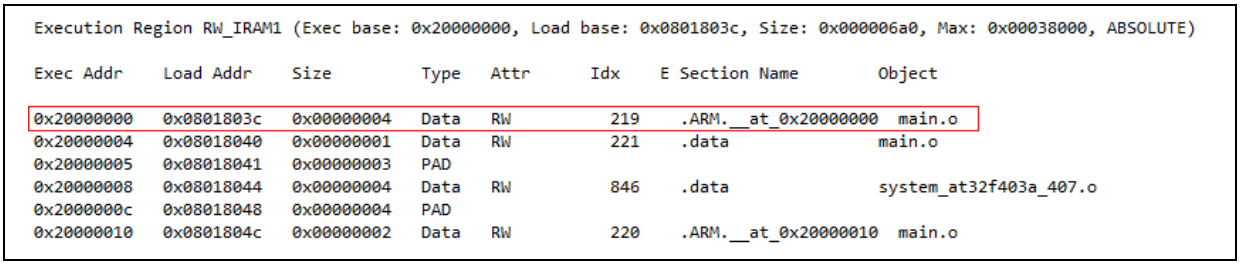


__Vectors是符号名，在这里的是向量表地址，0x08000000表示__Vectors链接符号所对应的地址。

4) 存储分布表

存储分布表详细罗列了每个节段的存储地址（加载地址）及执行地址和大小。其信息如下所示：

图 12. 存储分布



红色标记部分表示在目标文件main.o中指定为.ARM.__at_0x20000000节区的RW-data数据存储在ROM的0x0801803C地址，系统运行后将其初始化数据加载到RAM的0x20000000地址执行，长度为0x4字节。结合示例demo代码通俗的说法是，将main.c中指定了地址的rw_temp全部变量编译后其初始化值10存储在Flash地址的0x0801803C地址，运行后加载到内存的0x20000000地址进行使用。

通过map文件可知道，每个节段的存储地址及RW-data在RAM中的加载地址等严谨的分布信息。但这些加载地址都可以人为的通过在代码中使用（__attribute__）属性修改分散加载文件*.sct进行配

置。依据代码或数据的访问频率等信息，以此来进行更精细的控制，以达到对代码加载结构优化的目的。

4.2 分散加载的方法

要实现分散加载，Keil下有两种方式可以采用：一种是在code中使用attribute属性，主要针对于函数、数组和变量的分散加载来使用。另一种是修改.sct文件，此方法主要用在对整个目标文件进行分散加载。就灵活性、易读性和便利性考虑，推荐使用attribute属性来实现分散加载的方式。以下针对于函数、数组、变量和文件的分散加载进行分别说明。

demo请参考project\at_start_f403a\scatter。

方式一、将函数加载到指定位置

示例将main.c中的button_isr函数指定到0x08018000地址，可以在c文件中函数的定义处指定button_isr函数。

```
void button_isr(void) __attribute__ ((section(".ARM.__at_0x08018000")));
```

方式二、将数组加载到指定位置

```
uint8_t rw_data[2] __attribute__ ((section(".ARM.__at_0x20000010"))) = {0x1, 0x2};
```

方式三、将变量加载到指定位置

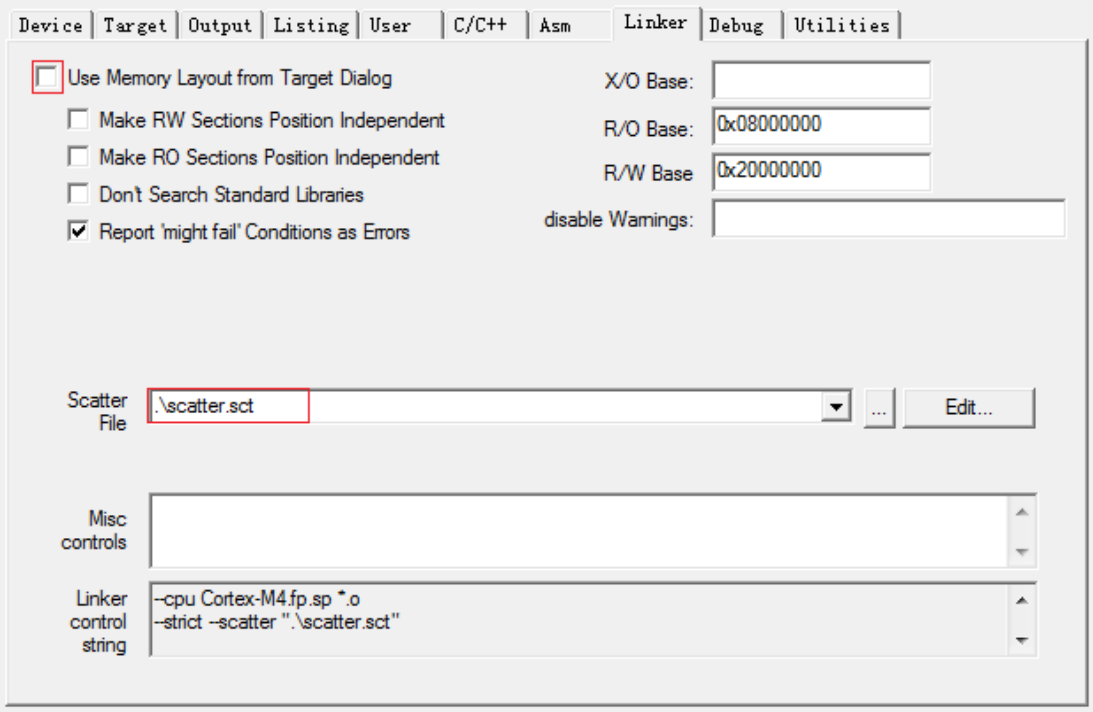
示例可以直接将c代码修改如下：

```
const uint32_t ro_temp __attribute__ ((section(".ARM.__at_0x08010000"))) = 10;      //RO
uint32_t rw_temp __attribute__ ((section(".ARM.__at_0x20000000"))) = 10;            //RW
```

方式四、将目标文件加载到指定位置

在keil中使用修改.sct文件进行分散加载需要对工程配置进行如下修改，修改.sct方法的详细使用请参考.sct的语法规则。

图 13. Keil 下 sct 文件编辑



其中Use Memory Layout from Target Dialog选项是默认勾选，应取消选择。点击Scatter File栏后的Edit对.sct文件进行编辑。示例将at32f403a_407_board.c的目标文件进行分散加载。

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x08000000 0x00020000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00020000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM1 0x20000000 0x00038000  {  ; RW data
   .ANY (+RW +ZI)
  }
}

LR_IROM2 0x08020000 0x00020000  {    ; load region size_region
  ER_IROM2 0x08020000 0x00020000  {  ; load address = execution address
   at32f403a_407_board.o (+RO)
  }
}
```

如上LR_IROM2区域表示将at32f403a_407_board.c编译生成的目标文件加载到0x08020000地址，LR_IROM2区域大小为0x00020000。通俗来说就是将at32f403a_407_board.c文件编译生成的目标文件链接到芯片内部flash的128KB区域之后，区域大小为128KB。

5 提高主频

提高主频的主要目的是提升MCU内核时钟频率及相应总线的时钟频率。因内核执行每条指令的时钟周期一定，频率的提升即可达到减少每条指令执行时间的目的。所以可以采用提高主频来提升系统执行性能。

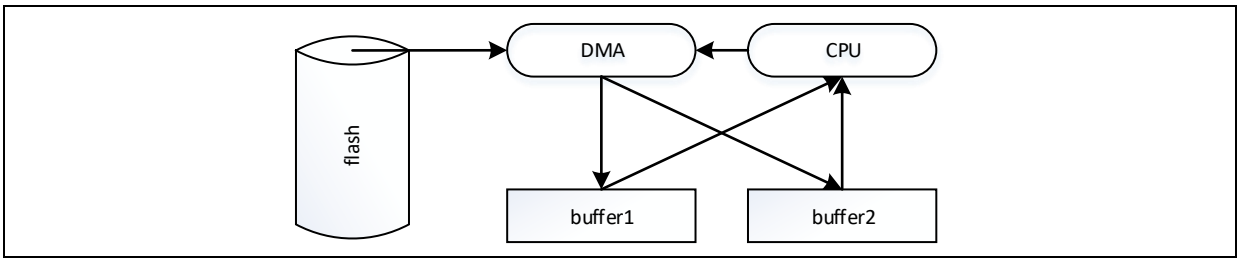
5.1 修改方法

详细配置方法请参考《AN0082_AT32F403A_407_CRM_Start_Guide_ZH》。

6 DMA 读取 Flash

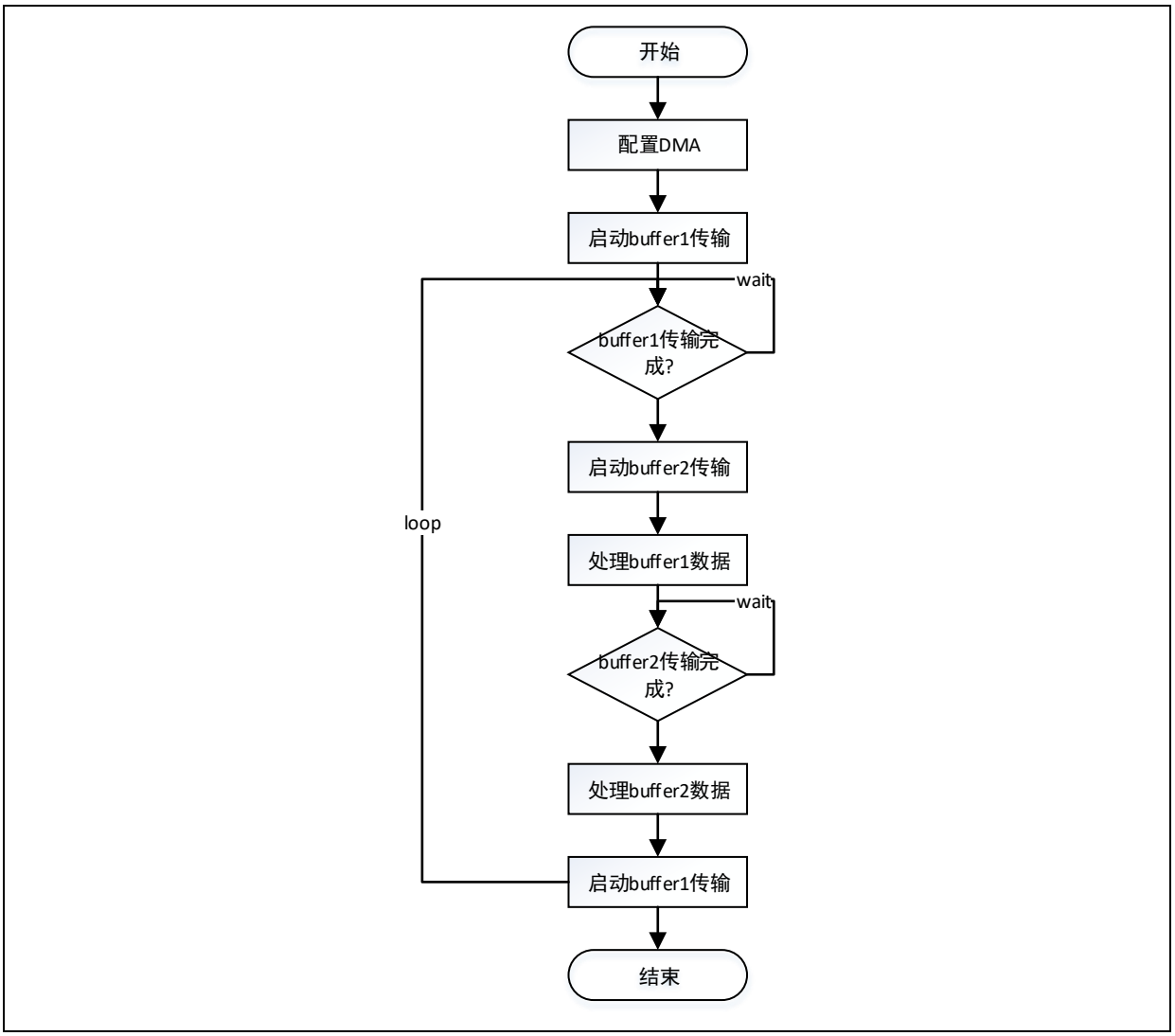
由前文可知部分AT32F系列MCU具有一定大小的零等待区域，在MCU上电启动时会自动从Flash的起始地址加载零等待区域大小的数据到零等待存储区，以待后续能快速读取使用。但Flash中存储地址在零等待映射范围以外的数据只能以Flash的访问流程进行读取访问，这样的方式相比于零等待区或SRAM的访问速度会变慢，而且读取数据越大越频繁越明显。针对以上问题，此章节提出以下方案和构想来实现Flash中零等待区映射范围以外数据的读取加速。构建模型如下图：

图 14. DMA 双 buffer 模型



为了尽量少的影响CPU的执行和正在进行的数据处理，Flash数据的读取工作交由DMA来进行，且采用双Buffer交替使用的方式。其大致的工作流程如下图：

图 15. DMA 双 buffer 处理流程



此模型的使用应特殊需求特殊处理。如Flash内数据块地址的指定，buffer大小的配置等，DMA的配

置demo可参考project\at_start_f403a\double_buffer。

7 版本历史

表 1. 文档版本历史

日期	版本	变更
2021.12.10	2.0.0	将内容及代码进版到V2

重要通知 – 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途（及其依据任何司法管辖区的法律的对应情况），或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：（A）对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；（B）航空应用；（C）航天应用或航天环境；（D）武器，且/或（E）其他可能导致人身伤害、死亡及财产损害的应用。如果采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险及法律责任仍将由采购商单独承担，且采购商应独力负责在前述应用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。