FAQ0086

常见问题解答

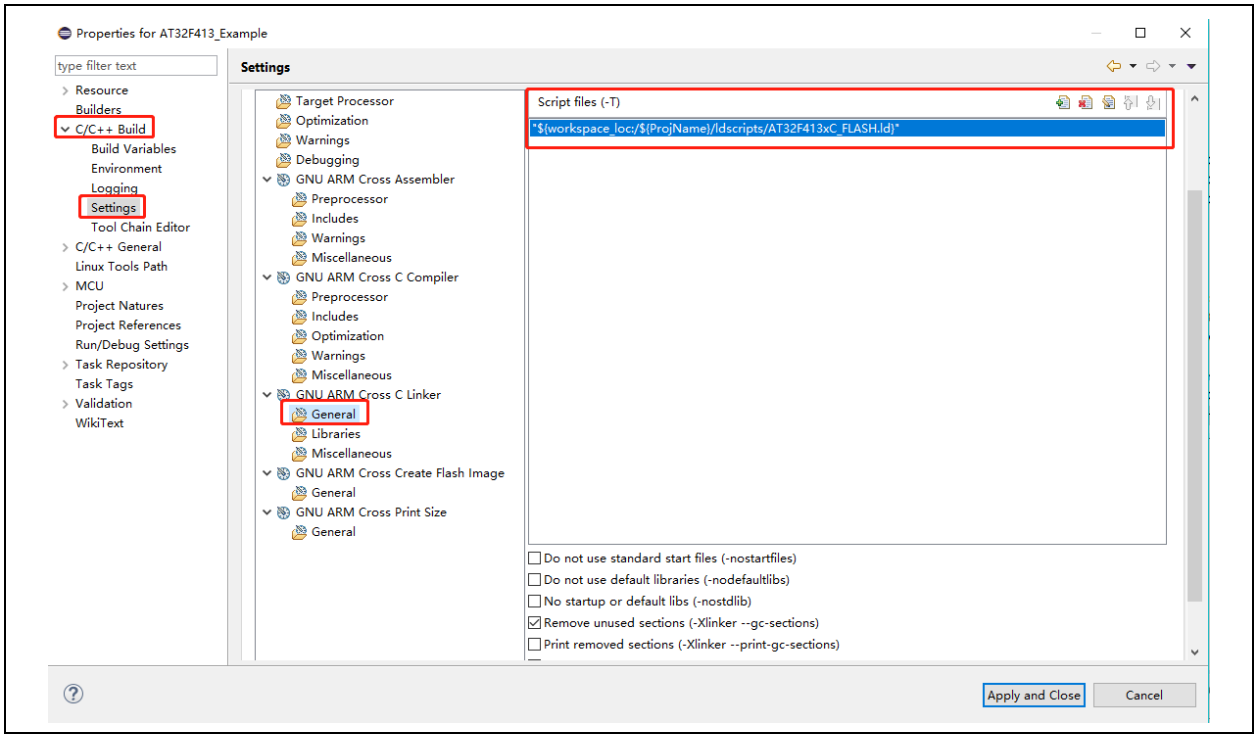
Eclipse中实现分散加载的方法

Questions：如何在 Eclipse 中实现分散加载？

Answer：

修改脚本链接文件可以将某些函数和数据编排到特定的区域内。

1. 链接脚本文件一般是放在根目录下的ldscripts 文件夹内，后缀为.ld。添加脚本链接文件的方法是“Project -> Properties -> C/C++ Build -> Setting -> Tool Settings -> “GNU ARM Cross C Linker” -> “General” -> 添加脚本链接文件。



2. 修改脚本链接文件将某些函数和数据编排到特定的区域内，脚本链接文件可以使用记事本打开。以AT32F413xC（FLASH=256K，SRAM=32K）为例，其划分区块默认如下：

```
/* Specify the memory areas */
MEMORY
{
FLASH (rx)        : ORIGIN = 0x08000000, LENGTH = 256K
RAM (xrw)          : ORIGIN = 0x20000000, LENGTH = 32K
}
```

r 代表 read-only，x 代表可执行代码，w 代表 read/write，ORIGIN 是该区块的起始地址，LENGTH 是该区块的大小。

如果需要将某些函数和数据编排到特定的 FLASH 区域内，则可以将 FLASH 划分为几个区域，以下是将 FLASH 划分为 3 个区域，可以将函数和数据编排到任意一个区域内。

```
/* Specify the memory areas */
```

```
MEMORY
{
FLASH_1 (rx)      : ORIGIN = 0x08000000, LENGTH = 128K
FLASH_2 (rx)      : ORIGIN = 0x08020000, LENGTH = 64K
FLASH_3 (rx)      : ORIGIN = 0x08030000, LENGTH = 64K
RAM (xrw)         : ORIGIN = 0x20000000, LENGTH = 32K
}
```

比如需要把算法文件 algorithm_1.c 和 algorithm_2.c 内的函数和数据编排到 FLASH_2 区域内；把算法文件 algorithm_3.c 和 algorithm_4.c 内的函数和数据编排到 FLASH_3 区域内。则需要在 SECTIONS 添加设置，如下红色部分。完整的脚本链接文件请参考附录 1。

```
/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH_1

  .algorithm_code1 :
  {
    . = ALIGN(4);
    *algorithm_1.o (.text .text*);
    *algorithm_2.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_2

  .algorithm_code2 :
  {
    . = ALIGN(4);
    *algorithm_3.o (.text .text*);
    *algorithm_4.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_3

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)                  /* .text sections (code) */
    *(.text*)                 /* .text* sections (code) */
     *(EXCLUDE_FILE (*algorithm_1.o * algorithm_2.o) .text .text*)
     *(EXCLUDE_FILE (*algorithm_3.o * algorithm_4.o) .text .text*)
    *(.glue_7)                /* glue arm to thumb code */
    *(.glue_7t)               /* glue thumb to arm code */
    *(.eh_frame)
    KEEP (*(.init))
    KEEP (*(.fini))
    . = ALIGN(4);
    _etext = .;               /* define a global symbols at end of code */
  } >FLASH_1
```

1) .algorithm_code1 和.algorithm_code2 是自行命名的 section 名称，用户在实际编写时可以自定义名称。 { } 包含的 .o 文件就是要放进 section 内的代码，{ } 末尾的 > FLASH_2 就是将 .algorithm_code1 这个 section 指定到先前定义的 FLASH_2 区块。例如 algorithm_1.o、algorithm_2.o 就是 algorithm_1.c 、algorithm_2.c 这两个 c 代码文档编译后的 object code， 写在这个 c 文档里的函数和数据，就全部会被编排到此 section 内。假设有 10 个函数，那 10 个函数就都会被放进来。

注意：

*1) .o 文件名前面都要加 * 号, 代表要将这个文件中的全部代码和数据都编排进来；*

2) .text 和 .text 是可执行的代码；*

3) *section 名称后的冒号与 section 名称之间要加空格，如：.algorithm_ code1 :。*

2) 将 .text{ } section 指定到 FLASH_1 区块。一定要加入 EXECLUDE_FILE 这个命令，使用 EXCLUDE_FILE 标示的代码就不会被编排到 FLASH_1，不然前面第 1)点所作的设置就会失效。此区段内的*（.text）和 *（.text*）， 就是告诉 linker 将 EXCLUDE_FILE 标示以外的代码都放到 .text 这个 section 内。

3. 对于 AT32F403/AT32F413/AT32F403A/AT32F407 等 FLASH 有零等待和非零等待的 MCU，如果需要将某些函数和数据编排到零等待或者非零等待域内，则可以将 FLASH 划分为 2 个区域，如下是以 AT32F413xC（FLASH=256K，SRAM=32K）为例，可以将函数和数据编排到任意一个区域内。

```
/* Specify the memory areas */
MEMORY
{
FLASH_ZW (rx)       : ORIGIN = 0x08000000, LENGTH = 96K
FLASH_NZW (rx)      : ORIGIN = 0x08018000, LENGTH = 160K
RAM (xrw)           : ORIGIN = 0x20000000, LENGTH = 32K
}
```

比如需要把对速率要求高的代码放到零等待，那么就可以将对速率要求不高的代码放到非零等待，留出零等待区存放对速率要求高的代码。比如将 nzw_1.c 和 nzw_2.c 内的函数和数据编排到 FLASH_NZW 区域内，则需要在 SECTIONS 添加设置，如下红色部分。完整的脚本链接文件请参考附录 2。

```
/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH_ZW

  .nzw_code :
  {
    . = ALIGN(4);
    *nzw_1.o (.text .text*);
    *nzw_2.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_NZW

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)                    /* .text sections (code) */
    *(.text*)                   /* .text* sections (code) */
    *(EXCLUDE_FILE (*nzw_1.o * nzw_2.o) .text .text*)
    *(.glue_7)                  /* glue arm to thumb code */
    *(.glue_7t)                 /* glue thumb to arm code */
    *(.eh_frame)
    KEEP (*(.init))
    KEEP (*(.fini))
    . = ALIGN(4);
    _etext = .;                 /* define a global symbols at end of code */
  } >FLASH_ZW
```

1) .nzw_code 是自行命名的 section 名称，用户在实际编写时可以自定义名称。 {} 包含的 .o 文件就

是要放进 section 内的代码，{ } 末尾的 > FLASH_NZW 就是将 .nzw_code 这个 section 指定到先前定义的 FLASH_NZW 区块。例如 nzw_1.o、nzw_2.o 就是 nzw_1.c 、nzw_2.c 这两个 c 代码文档编译后的 object code， 写在这个 c 文档里的函数和数据，就全部会被编排到此 section 内。假设有 10 个函数，那 10 个函数就都会被放进来。

注意:

1) *.o 文件名前面都要加 * 号, 代表要将这个文件中的全部代码或数据都编排进来;*
2) *.text 和 .text* 是可执行的代码;*
3) *section 名称后的冒号与 section 名称之间要加空格，如：.nzw_code :。*

2) 将 .text{ } section 指定到 FLASH_ZW 区块。一定要加入 EXECLUDE_FILE 这个命令，使用 EXCLUDE_FILE 标示的代码就不会被编排到 FLASH_ZW，不然前面第 1)点所作的设置就会失效。此区段内的*（.text）和 *（.text*）， 就是告诉 linker 将 EXCLUDE_FILE 标示以外的代码都放到 .text 这个 section 内。

附录 1

```
/*
*****************************************************************************
**
**  File        : AT32F413xC_FLASH.ld
**
**  Abstract    : Linker script for AT32F413xC Device with
**                256KByte FLASH, 32KByte RAM
**
**                Set heap size, stack size and stack location according
**                to application requirements.
**
**                Set memory bank area and size if external memory is used.
**
**  Target      : Artery Tek AT32
**
**  Environment : Arm gcc toolchain
**
*****************************************************************************
*/
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20008000;      /* end of RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;          /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
FLASH_1 (rx)      : ORIGIN = 0x08000000, LENGTH = 128K
FLASH_2 (rx)      : ORIGIN = 0x08020000, LENGTH = 64K
FLASH_3 (rx)      : ORIGIN = 0x08030000, LENGTH = 64K
RAM (xrw)           : ORIGIN = 0x20000000, LENGTH = 32K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
```

```
    . = ALIGN(4);
  } >FLASH_1

  .algorithm_code1 :
  {
    . = ALIGN(4);
    *algorithm_1.o (.text .text*);
    *algorithm_2.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_2

  .algorithm_code2 :
  {
    . = ALIGN(4);
    *algorithm_3.o (.text .text*);
    *algorithm_4.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_3

  /* The program code and other data goes into FLASH */
  .text :
  {
    . = ALIGN(4);
    *(.text)                    /* .text sections (code) */
    *(.text*)                   /* .text* sections (code) */
    *(EXCLUDE_FILE (*algorithm_1.o *algorithm_2.o) .text .text*)
    *(EXCLUDE_FILE (*algorithm_3.o *algorithm_4.o) .text .text*)
    *(.glue_7)                  /* glue arm to thumb code */
    *(.glue_7t)                 /* glue thumb to arm code */
    *(.eh_frame)
    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    _etext = .;                 /* define a global symbols at end of code */
  } >FLASH_1

  /* Constant data goes into FLASH */
  .rodata :
  {
    . = ALIGN(4);
    *(.rodata)                  /* .rodata sections (constants, strings, etc.) */
    *(.rodata*)                 /* .rodata* sections (constants, strings, etc.) */
    . = ALIGN(4);
  } >FLASH_1

  .ARM.extab    : { *(.ARM.extab* .gnu.linkonce.armextab.*) } >FLASH_1
  .ARM : {
    __exidx_start = .;
    *(.ARM.exidx*)
    __exidx_end = .;
  } >FLASH_1

  .preinit_array     :
  {
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(.preinit_array*))
    PROVIDE_HIDDEN (__preinit_array_end = .);
  } >FLASH_1

  .init_array :
  {
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
```

```
    KEEP (*(.init_array*))
    PROVIDE_HIDDEN(__init_array_end = .);
  }>FLASH_1

  .fini_array:
  {
    PROVIDE_HIDDEN(__fini_array_start = .);
    KEEP (*(SORT(.fini_array.*)))
    KEEP (*(.fini_array*))
    PROVIDE_HIDDEN(__fini_array_end = .);
  }>FLASH_1

  /* used by the startup to initialize data */
  _sidata = LOADADDR(.data);

  /* Initialized data sections goes into RAM, load LMA copy after code */
  .data :
  {
    . = ALIGN(4);
    _sdata = .;            /* create a global symbol at data start */
    *(.data)               /* .data sections */
    *(.data*)              /* .data* sections */

    . = ALIGN(4);
    _edata = .;            /* define a global symbol at data end */
  }>RAM AT> FLASH_1

  /* Uninitialized data section */
  . = ALIGN(4);
  .bss :
  {
    /* This is used by the startup in order to initialize the .bss secion */
    _sbss = .;             /* define a global symbol at bss start */
    __bss_start__ = _sbss;
    *(.bss)
    *(.bss*)
    *(COMMON)

    . = ALIGN(4);
    _ebss = .;             /* define a global symbol at bss end */
    __bss_end__ = _ebss;
  }>RAM

  /* User_heap_stack section, used to check that there is enough RAM left */
  ._user_heap_stack:
  {
    . = ALIGN(8);
    PROVIDE ( end = . );
    PROVIDE ( _end = . );
    . = . + _Min_Heap_Size;
    . = . + _Min_Stack_Size;
    . = ALIGN(8);
  }>RAM



  /* Remove information from the standard libraries */
  /DISCARD/ :
  {
    libc.a ( * )
    libm.a ( * )
    libgcc.a ( * )
  }
```

```
  .ARM.attributes 0 : { *(.ARM.attributes) }
}
```

附录 2

```
/*
*****************************************************************************
**
**  File        : AT32F413xC_FLASH.ld
**
**  Abstract    : Linker script for AT32F413xC Device with
**                256KByte FLASH, 32KByte RAM
**
**                Set heap size, stack size and stack location according
**                to application requirements.
**
**                Set memory bank area and size if external memory is used.
**
**  Target      : Artery Tek AT32
**
**  Environment : Arm gcc toolchain
**
*****************************************************************************
*/

/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20007FFF;    /* end of RAM */

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
FLASH_ZW (rx)      : ORIGIN = 0x08000000, LENGTH = 96K
FLASH_NZW (rx)     : ORIGIN = 0x08018000, LENGTH = 160K
RAM (xrw)          : ORIGIN = 0x20000000, LENGTH = 32K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH_ZW

  .nzw_code :
  {
    . = ALIGN(4);
    *nzw_1.o (.text .text*);
    *nzw_2.o (.text .text*);
    . = ALIGN(4);
  } > FLASH_NZW

  /* The program code and other data goes into FLASH */
  .text :
```

```
{
   . = ALIGN(4);
   *(.text)                /* .text sections (code) */
   *(.text*)               /* .text* sections (code) */
   *(EXCLUDE_FILE (*nzw_1.o *nzw_2.o) .text .text*)
   *(.glue_7)              /* glue arm to thumb code */
   *(.glue_7t)             /* glue thumb to arm code */
   *(.eh_frame)

   KEEP (*(.init))
   KEEP (*(.fini))

   . = ALIGN(4);
   _etext = .;             /* define a global symbols at end of code */
} >FLASH_ZW

/* Constant data goes into FLASH */
.rodata :
{
   . = ALIGN(4);
   *(.rodata)              /* .rodata sections (constants, strings, etc.) */
   *(.rodata*)             /* .rodata* sections (constants, strings, etc.) */
   . = ALIGN(4);
} >FLASH_ZW

.ARM.extab    : { *(.ARM.extab* .gnu.linkonce.armextab.*) } >FLASH_ZW
.ARM : {
   __exidx_start = .;
   *(.ARM.exidx*)
   __exidx_end = .;
} >FLASH_ZW

.preinit_array     :
{
   PROVIDE_HIDDEN (__preinit_array_start = .);
   KEEP (*(.preinit_array*))
   PROVIDE_HIDDEN (__preinit_array_end = .);
} >FLASH_ZW
.init_array :
{
   PROVIDE_HIDDEN (__init_array_start = .);
   KEEP (*(SORT(.init_array.*)))
   KEEP (*(.init_array*))
   PROVIDE_HIDDEN (__init_array_end = .);
} >FLASH_ZW
.fini_array :
{
   PROVIDE_HIDDEN (__fini_array_start = .);
   KEEP (*(SORT(.fini_array.*)))
   KEEP (*(.fini_array*))
   PROVIDE_HIDDEN (__fini_array_end = .);
} >FLASH_ZW

/* used by the startup to initialize data */
_sidata = LOADADDR(.data);

/* Initialized data sections goes into RAM, load LMA copy after code */
.data :
{
   . = ALIGN(4);
   _sdata = .;             /* create a global symbol at data start */
   *(.data)                /* .data sections */
   *(.data*)               /* .data* sections */
```

```
    . = ALIGN(4);
    _edata = .;          /* define a global symbol at data end */
  } >RAM AT> FLASH_ZW

  /* Uninitialized data section */
  . = ALIGN(4);
  .bss :
  {
    /* This is used by the startup in order to initialize the .bss secion */
    _sbss = .;           /* define a global symbol at bss start */
    __bss_start__ = _sbss;
    *(.bss)
    *(.bss*)
    *(COMMON)

    . = ALIGN(4);
    _ebss = .;           /* define a global symbol at bss end */
    __bss_end__ = _ebss;
  } >RAM

  /* User_heap_stack section, used to check that there is enough RAM left */
  ._user_heap_stack:
  {
    . = ALIGN(4);
    PROVIDE ( end = . );
    PROVIDE ( _end = . );
    . = . + _Min_Heap_Size;
    . = . + _Min_Stack_Size;
    . = ALIGN(4);
  } >RAM

  /* Remove information from the standard libraries */
  /DISCARD/ :
  {
    libc.a ( * )
    libm.a ( * )
    libgcc.a ( * )
  }

  .ARM.attributes 0 : { *(.ARM.attributes) }
}
```

类型： 开发工具

适用型号： AT32F4 全系列

主功能： 无

次功能： 无

文档版本历史

日期	版本	变更
2022.2.23	2.0.0	最初版本

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，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) 汽车应用或汽车环境；(D) 航天应用或航天环境，且/或(E) 武器。因雅特力产品不是为前述应用设计的，而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，风险由购买者单独承担，并且独力负责在此类相关使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。