目录

1. 绪论

在编译一个程序时，最后是将每个输出文件链接在一起，最后一步就是运行“.ld”文件内容。每一个链接过程都由链接脚本控制，链接脚本主将定义的 **section** 对与文件内的输出文件读取，合并，生成目标文件。

通过对脚本的调整可以实现特殊应用的程序布局实现，如 bootloader 双应用程序开发，自带部分升级特性的指定函数段地址，或差异化应用的 RAM 变量指定地址。

KF32 输出文件格式为 elf32。Ram 空间设计部分地址空间为低功耗唤醒不丢失数据区域，如前面 16K，具体应以产品规格书为准，即变型使用 startup 和差异化唤醒的 startup2[控制 for 循环跳过.lpdata 段].

默认情况下程序代码函数段归属为 flash 空间下的.text，全局初始化常量归属为 flash 空间下. **r**data 段,局部初始化常量归属为 flash 空间下.rodata 段。全局初始化变量归属为 ram 空间下.data,全局未初始化变量归属为 ram 空间下.bss 或 COMM。另外 C++类的全局初始化与和初始化列表归属为 flash 空间下的.init 和.init_array。

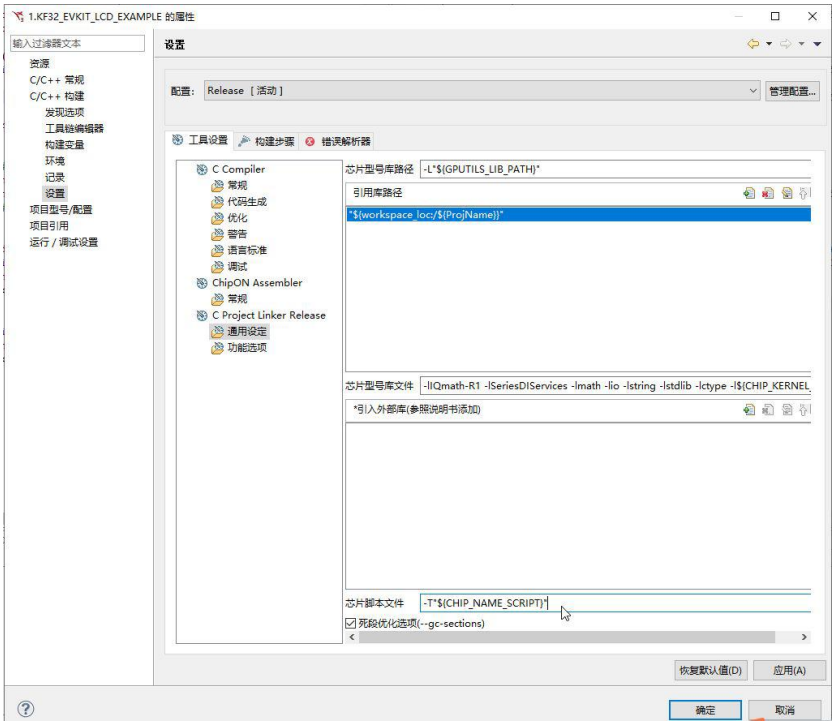
脚本中具有=运算中的.为语法格式的当前偏移量地址。段配置格式为 file(内容段),其中*为通配符,如*(.text*)对应任意文件的代码 text 起始段。*vector.o(.text*)对应编译输出对象 vector.o 文件下的代码 text 起始段。

__text_end__ 、__data_start__、__data_end__、__bss_start__、__bss_end__、__allot_end__、__initial_sp 等为必须的脚本符号，该符号构成工具实现的组成部分。其中__data_start__必须为映射到 ram 的 data 起始部分，若需要保留 RAM 的前 N 空间，可以通过修改 MEMORY 下的 ram　　：ORIGIN = 0x10000000, LENGTH = 0x00004000 实现偏移和保留后的 ram 可用空间长度。

程序文件的生成会将 ram 段的全局初始化部分附加在 flash 段的__text_end__位置后面空间，即代码与全局数据构成完整的程序内容。

2. 使用自定义的.ld 文件

编译过程，IDE 默认从安装目录下获取.ld 文件。默认的路径通过工程属性->C/C++ 构建->C Link 是如图，其中"${CHIP_NAME_SCRIPT}"为工具变量：



默认的**.ld** 文件在安装目录下的：ChipONIDE/KungFu32/ChiponCC32/[选择工具链，如 ccrl_issue]/scripting /[项目型号如 **KF32L530MNS**].ld。

将该路径的对应的项目型号相关的 ld 文件复制到工程路径下，更改芯片脚本文件为：-T"${ProjDirPath}/${CHIP_NAME}.ld"，或-T"../[脚本文件名如 **KF32L530MNS**].ld"。

其中${X}为 IDE 变量引用格式，如字面意思 **ProjDirPath 的项目所在路径和 CHIP_NAME 的项目使用型号。**

3. 内存地址说明

如 KF32L530MNS 的内存拥有 512K 的 Flash 和 128 的 RAM。Flash 的起始地址为 0x0000 0000；RAM 的起始地址为 0x1000 0000。链接文件中根据型号资源配置 Flash 和 RAM 的起始地址及长度。脚本下内存配置如下：

```
MEMORY
{
  flash       : ORIGIN  =  0x00000000, LENGTH = 0x00010000
  ram         : ORIGIN  =  0x10000000, LENGTH = 0x00004000
  da_mem      : ORIGIN  =  0x0C001C00, LENGTH = 0x00000400
  mode_mem    : ORIGIN  =  0x0C001800, LENGTH = 0x00000004
  pro_mem     : ORIGIN  =  0x0C001000, LENGTH = 0x00000004
  ee_mem      : ORIGIN  =  0x7F000000, LENGTH = 0x00001000
  config_mem1 : ORIGIN  =  0x31000000, LENGTH = 0x00000010
  config_mem2 : ORIGIN  =  0x31000010, LENGTH = 0x00000010
}
```

编程相关空间为代码 **flash** 和变量 **ram**，其他空间为辅助的定义 **hex** 文件内容下地址偏移与大小,如 **da_mem** 的 **flash date** 区域，**mode_mem** 的模式配置区域，**pro_mem** 的加密模式区域，其他保留或扩展。

脚本文件下布局控制实现段配置格式如下，详细见文件与控制描述部分。

```
SECTIONS
{
  .text :
```

```
    {
      …
    } > flash
    .data :
    {
      …
    } > ram
    …
  }
```

注：在定义 **flash** 或 **ram** 段时，段名结尾必须以*为结束，代码部分不需要。即同名合并原则下，工具给予每个函数或变量段后缀从而确定唯一，链接器默认开启死段优化，即未使用代码的不分配空间,同时意味着不能配置相同前缀的段，如 **A***与 **AAA***将视为相同。针对资源赋值可以使用综合的数据表达，如 **0x00010000**、**64K**、**2K+192**，在 **SECTIONS** 下的段控制 **. = 0xXXX**(其中“=”前后必须使用空格间隔)的值为基于当前的偏移，即非目标区域的绝对地址，该最终值由链接器基于 **MEMORY** 的起始值计算实现。

若指定地址的分割空间，可以指定填充值，如在 **flash** 填充区域设置填充值 **0x12345678**，则修改对应 **text** 段格式如下：

```
    .text :
    {
      …
    } > flash =0x12345678
```

4. 定义常量在指定 Flash 地址

在应用中若需要将常量指定 Flash 地址，可以在.text 段进行段定位。在.text 前 512 字节需要预留给向量表存放，不可占用【即芯片启动机制：中断向量表与初始 SP 和程序地址入口】。

```
  .text :
  {
    . = 0x0000;
    KEEP (*vector.o(.text*)) /* chip interrupt vector ,writed in file named vector.s or vector.c */
```

当前工具设计向量表使用独立的文件，即 **vector.c** 或 **vector.asm**.因此设计了该文件代码 **obj** 优先存放,并对应的代码.**text** 段存放的实现中断向量表入口。另外保存了对象名字为**_start** 的与脚本指定入口 **ENTRY(_start)**一致。【其中 **KEEP** 关键字修饰指明不参与死段优化，若向量表混合到其他文件或同文件下具有多个 **text** 段时，应调整脚本使能一致，如 **KEEP (*(.vector*))**并向量表内容段使用`__attribute__((section(".vector")))`修饰】。

示例定义常量参数代码的起始段，如定义在向量表后的 **0x0000 0200** 处。修改脚本为：

```
    . = 0x0000;
    KEEP (*vector.o(.text*))    /* chip interrupt vector ,writed in file named vector.s or vector.c */

    . = (. + 3) & (-4);  /* align 4 */
    __vec_end__ = .;
    . = 0x200;              /* after and befor '=' must space by write */
    KEEP (*(.ConstData*)) /* define code section function ,must end with '*' */
    *(.text*)                   /* default function code space */
```

代码编写使用 **section** 关键字指定,如

```
__attribute__((section(".ConstData"))) uint8 Test[] = "12345678";
```

编译通过后可以通过**.map** 文件或者 **HEX** 文件查看定义的位置是否生效。

若定义在代码的结束段定义，需要保证不覆盖代码，示例代码量小于等于 0x4000，修改脚本实现常量信息存放在 0x4000：

```
*(.userrodata*)
. = 0x4000;          /* after and befor '=' must space by write */
KEEP (*(.ConstData*)) /* define code section function ,must end with '*' */
. = (. + 3) & (-4);
__text_end__ = .;
                              /*record: global variable value of initialization */
} > flash
```

5. 定义变量及函数在指定 RAM 空间

RAM 在上电时内容是随机的，使用前需要将拥有初值的变量赋值。如在"./config/vector.c"文件中规定了单片机从"startup"函数开始运行，"startup"函数的作用是给拥有初值的变量赋值。即在运行 main 之前，将变量初始化完毕。

在应用中若需要将指定 RAM 地址，即在.data 中定义需要用户段".XXXX"，并指定其偏移地址。示例定义在 0x10001000 的 UserRAMData 修改脚本如下：

```
*(.indata*)          /* server space for ram function */
*(.inrdata*)
*(.inrodata*)

. = 0x1000;
KEEP(*(.UserRAMData*))
*(.UserRAM2Data*)

. = (. + 3) & (-4);
*(.data*)            /* global variable with  initialization */

. = (. + 3) & (-4);
__data_end__ = .;
```

代码部分追加属性修饰指定段，如 `__attribute__((section(".UserRAMData")))` `uint8 Testram[] = "12345678";`编译通过后可以通过**.map** 文件或者 反汇编 **lst** 文件查看定义的位置是否生效。

NOTE：需要注意的事，编译处理变量默认段为.data、.bss 或 COMMON，并且同属性合并连续存放，__data_end__与__bss_start__应连续，即中间不应插入自定义段。__data_start__与__data_end__之间的内容作为全局带初始化识别，即若区间指定段的无初始化默认内容为 0.该部分数据内容会附加到 flash 的__text_end__标明空间后面。

NOTE：设计在__data_end__之后的固定地址段变量应该代码编写为初始化的全局变量(初始化值会丢失)，设计在__data_start__与__data_end__之间的指定地址段应该将地址分配在前面，即顺序使用安排，若不连续并需要建立在靠后，建议地址与实际需求结尾地址相近并被.data 段间隔的 data 空间应能满足默认全局变量空间需求(否则基于__data_start__与__data_end__规则填充 0 占用额外占用 flash 空间存放与增长初始化时间）。

6. 低功耗模式下保持数据应用

单片机的 SRAM 分为主要分为两个区域。一部分 LPRAM 在 STANDBY 及 STOP1 的低功耗模式仍然可以保持，为 SRAM 区域的前 16K 【规格型号下的】，另一部分为通用 RAM。休眠前，将 PM_CTL0 的 bit19 置 1，即可保持数据。在 SRAM 的前 16K 空间中规划出 section 段，将要保持的变量放置于该段，该变量的数据可以在 STANDBY 及 STOP1 的低功耗模式下保持。

按照正常的启动逻辑，单片机在复位后会先执行"startup"函数，执行此函数会将变量进行初始化。在 STANDBY 模式及 STOP1 模式下，唤醒后代码从头运行。

若需要低功耗下保持数据，需要避免每次复位处调用默认"startup"函数，需要更改启动逻辑，启动后从"main"开始运行【即修改向量表文件替换 startup 为 main】。main 函数判断是否需要初始调用默认 startup 或调用变型的 startup 函数，如 startup1，基于 startup 修改内容：

```
while(begin < end)
{
    *begin++ = *s++;
}
```

为

```
extern unsigned int __lpdata_start__;  // 脚本变量
extern unsigned int __lpdata_end__;    // 脚本变量

while(begin < end)
{
    if(begin == ( unsigned int * )(&__lpdata_start__))
        begin = ( unsigned int * )(&__lpdata_end__);
    *begin++ = *s++;
}
```

变量使用__attribute__((section(".lpdata")))修饰，默认脚本为

```
    KEEP (*(.ramvector*))      /* default server for vector writed in ram */

  __lpdata_start__ = (. + 3) & (-4);
    *(.lpdata*)            /* Reset but Keep Space  */
    /* . = 0x4000 ;  */   /* Max length limit is 16K byte */
  __lpdata_end__ =  (. + 3) & (-4);

    . = (. + 3) & (-4);
    *(.indata*)        /* server space for ram function */
    *(.inrdata*)
    *(.inrodata*)

    . = (. + 3) & (-4);
    *(.data*)          /* global variable with  initialization */
```

若低功耗保持数据空间需求明显小于空间，即根据 ram 布局顺序 lpdata 段实现在前 16K 空间内容，若数据空间需求量接近或可能接近下，可调整脚本为：

```
    *(.lpdata*)            /* Reset but Keep Space  */
     . = 0x4000 ;      /* Max length limit is 16K byte */
```

即使能__lpdata_end 为 0x4000 的偏移空间占用，lpdata 未使用空间将默认填充为 0 的假定强制使用，也可以根据使用量设计如前 1K 空间作为 lpdata 区域的配置值 0x400。

7. bootloader 双程序开发模式应用

复制型号模板 ld 文件到项目中，修改链接脚本为项目下的对应 ld 文件。示例 boot 空间为 0x4000 并 app 起始地址空间为 0x4000[向量表保持在起始地址位置]下的修改脚本内容：

```
MEMORY
{
  flash       : ORIGIN =  0x00004000, LENGTH = 0x00010000-0x00004000
  ram         : ORIGIN =  0x10000000, LENGTH = 0x00004000
```

修改起始地址和长度实现 boot 与 app 的程序空间划分，其他内容无特殊需求下保持不变更。

其中boot程序引导进入app时，根据app脚本所在向量表位置实现向量表重映射与代码跳转

```
#define ApplicationAddress 0x00004000
typedef  void (*pFunction)(void);
pFunction Jump_To_Application;
volatile uint32_t JumpAddress;
void __set_MSP(uint32_t value){
    asm(" MOV SP,%0 \n": :"r"(value): "sp");
}
void SYSCTL_Vector_Offset_Config (uint32_t VectorOffset){
    SYS_VECTOFF = VectorOffset;
}
{
        asm ("DSI"); // 根据需要自行关闭 boot 中断源外设
        SYSCTL_Vector_Offset_Config(0x4000);
        JumpAddress = *(__IO uint32_t*) (ApplicationAddress + 4);
        Jump_To_Application = (pFunction) JumpAddress;
        __set_MSP(*(__IO uint32_t*) ApplicationAddress);
        Jump_To_Application();
}
```

8. 版本历史

序号	版本	变更描述	影响范围	日期
1	V1.0	初稿	无	2021-2-18
2	V1.1	修正低功耗保持数据应用介绍的初始化表达错误 修正低功耗保持 ram 变量属性声明段名".lpdata*"为不带*的".lpdata" 添加内存地址说明部分的脚本中名需要以*为原因的介绍	原表达初始化功能异常	2021-6-1
3	V1.2	修正 RAM 定义的描述的自定义段()错误，仅 KEEP 修饰下需要；增加脚本常量值支持灵活的书写。	原格式书写链接阶段脚本语法错误	2022-3-22