# 目 录

[目 录 I](#_Toc502839771)

[1 C语言基础 1](#_Toc502839772)

[1.1 结构体成员乱序初始化 1](#_Toc502839773)

[2 相关格式 3](#_Toc502839774)

[2.1 编号 3](#_Toc502839775)

[2.1.1 1-9）版 3](#_Toc502839776)

[2.1.2 1-10）版 3](#_Toc502839777)

[2.1.3 （1-10）版 4](#_Toc502839778)

[2.1.4 1. 代码版 4](#_Toc502839779)

[2.2 图表 5](#_Toc502839780)

[2.2.1 技术 5](#_Toc502839781)

[参考文献 7](#_Toc502839782)

# 基础

* 1. Data Store异常处理

如果Security Data读UECC，那么后端应返回Empty Data，使得盘能够正常工作；但是对于其他配置信息，后端可以返回之前的备份数据。

* 1. SATA DM测试

PwCycle\_RdmCmds和PwCycle\_DataCmp测试脚本只看GoodData和DNR（Device Not Ready）三项。

* 1. shadow register

有阴影的寄存器，表示在物理上这个寄存器对应2个寄存器，一个是程序员可以写入或读出的寄存器，称为preload register(预装载寄存器)，另一个是程序员看不见的、但在操作中真正起作用的寄存器，称为shadow register(影子寄存器)

设计preload register和shadow register的好处是，所有真正需要起作用的寄存器(shadow register)可以在同一个时间(发生更新事件时)被更新为所对应的preload register的内容，这样可以保证多个通道的操作能够准确地同步。如果没有shadow register，或者preload register和shadow register是直通的，即软件更新preload register时，同时更新了shadow register，因为软件不可能在一个相同的时刻同时更新多个寄存器，结果造成多个通道的时序不能同步，如果再加上其它因素(例如中断)，多个通道的时序关系有可能是不可预知的。

这样做可以加快程序的相应和信息的处理。

其中ARM的FIQ有用到，也就是fast interrupt，快速中断的意思。

因为FIQ是有Shadow Register的，所以你在使用它或者是设置它的时候，指令可以先存在Shadow Register里面，等到下一个硬件周期就会立即执行。

而没有Shadow Register的IRQ（interrupt），就要等Register里面的数据执行完毕后才能写入，那样就稍慢一些。

这也就是FIQ比IRQ快的一个原因。

* 1. Keil SCT分散加载文件

分散加载机制允许为链接器指定映像的存储器映射信息，可实现对映像组件分组和布局的全面控制。分散加载能够将加载和运行时存储器中的代码和数据描述在被称为分散加载描述文件的一个文本描述文件中，以供连接时使用。

（1）分散加载区

分散加载区域分为两类：

加载区：该映像文件开始运行前存放的区域，即当系统复位和加载时的代码和数据。

执行区：映像文件运行时的区域，即系统启动后，应用程序进行执行和数据访问的存储器区域，应用程序启动过程中，从每个加载区可创建一个或多个执行区。

映象中所有的代码和数据准确地分为一个加载区和一个执行区。

（2）分散加载文件示例

ROM\_LOAD 0x0000 0x4000

{

ROM\_EXEC 0x0000 0x4000; Root region

{

\* (+RO); All code and constant data

}

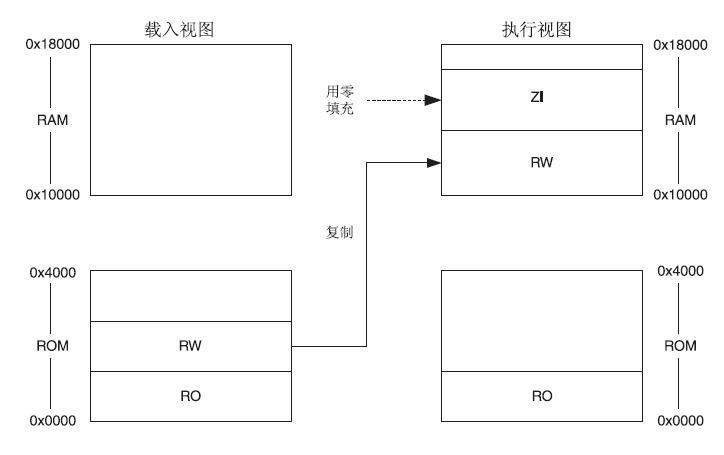
RAM 0x10000 0x8000

{

\* (+RW, +ZI); All non-constant data

}

}



load\_region\_name start\_address | "+"offset [attributes] [max\_size]

{

execution\_region\_name start\_address | "+"offset [attributes][max\_size]

{

module\_select\_pattern ["("

("+" input\_section\_attr | input\_section\_pattern)

([","] "+" input\_section\_attr | "," input\_section\_pattern)) \*

")"]

}

}

load\_region： 加载区，用来保存永久性数据（程序和只读变量）的区域；

execution\_region： 执行区，程序执行时，从加载区域将数据复制到相应执行区后才能被正确执行；

load\_region\_name： 加载区域名，用于“Linker”区别不同的加载区域，最多31个字符；

start\_address： 起始地址，指示区域的首地址；

+offset： 前一个加载区域尾地址＋offset 做为当前的起始地址，且“offset”应为“0”或“4”的倍数；

attributes： 区域属性，可设置如下属性：

PI 与地址无关方式存放；

RELOC 重新部署，保留定位信息，以便重新定位该段到新的执行区；

OVERLAY 覆盖，允许多个可执行区域在同一个地址，ADS不支持；

ABSOLUTE 绝对地址（默认）；

max\_size： 该区域的大小；

execution\_region\_name：执行区域名；

start\_address： 该执行区的首地址，必须字对齐；

+offset： 同上；

attributes： 同上；

PI 与地址无关，该区域的代码可任意移动后执行；

OVERLAY 覆盖；

ABSOLUTE 绝对地址（默认）；

FIXED 固定地址；

UNINIT 不用初始化该区域的ZI段；

module\_select\_pattern： 目标文件滤波器，支持通配符“\*”和“?”；

\*.o匹配所有目标，\* （或“.ANY”）匹配所有目标文件和库。

input\_section\_attr： 每个input\_section\_attr必须跟随在“＋”后；且大小写不敏感；

RO-CODE 或 CODE

RO-DATA 或 CONST

RO或TEXT, selects both RO-CODE and RO-DATA

RW-DATA

RW-CODE

RW 或 DATA, selects both RW-CODE and RW-DATA

ZI 或 BSS

ENTRY, that is a section containing an ENTRY point.

FIRST，用于指定存放在一个执行区域的第一个或最后一个区域；

LAST，同上；

input\_section\_pattern： 段名；

汇编中指定段：

AREA vectors, CODE, READONLY

C中指定段：

#pragma arm section [sort\_type[[=]"name"]] [,sort\_type="name"]\*

sort\_type: code、rwdata、rodata、zidata

如果“sort\_type”指定了但没有指定“name”，那么之前的修改的段名将被恢复成默认值。

#pragma arm section // 恢复所有段名为默认设置。

应用：

#pragma arm section rwdata = "SRAM",zidata = "SRAM"

static OS\_STK SecondTaskStk[256]; // “rwdata”“zidata”将定位在“sram”段中。

#pragma arm section // 恢复默认设置

（4）程序中对区域地址引用的方法

Load$$region\_name$$Base Load address of the region.

Image$$region\_name$$Base Execution address of the region.

Image$$region\_name$$Length Execution region length in bytes (multiple of 4).

Image$$region\_name$$Limit Address of the byte beyond the end of the execution region.

Image$$region\_name$$ZI$$Base Execution address of the ZI output section in this region.

Image$$region\_name$$ZI$$Length Length of the ZI output section in bytes (multiple of 4).

Image$$region\_name$$ZI$$Limit Address of the byte beyond the end of the ZI output sectionin the execution region.

SectionName$$Base Input Address of the start of the consolidated section called SectionName.

SectionName$$Limit Input Address of the byte beyond the end of the consolidated section called SectionName.

Load： 加载区，即存放地址；

Image： 执行区，即运行地址；

Base： 区首地址；

Limit： 区尾地址；

Length： 区长度；

region\_name： RO、RW、ZI、load\_region\_name、execution\_region\_name；

例如：

“RAM1”区域的首地址： Image$$RAM1$$Base

上例中“sram”段首地址： sram$$Base

汇编引用示例：

IMPORT |Load$$Exec\_RAM1$$Base| // Exec\_RAM1 为“RW”段

IMPORT |Image$$Exec\_RAM1$$Base|

IMPORT |Image$$Exec\_RAM1$$Length|

IMPORT |Image$$Exec\_RAM1$$Limit|

LDR R0, =|Load$$Exec\_RAM1$$Base|

LDR R1, =|Image$$Exec\_RAM1$$Base|

LDR R2, =|Image$$Exec\_RAM1$$Limit|

0

CMP R1, R2

LDRCC R3, [R0], #4

STRCC R3, [R1], #4

BCC %b0

C 引用：

extern unsigned char Load$$Exec\_RAM1$$Base**;**

extern unsigned char Image$$Exec\_RAM1$$Base**;**

extern unsigned char Image$$Exec\_RAM1$$Length**;**

void MoveRO**(**void**)**

**{**

unsigned char **\*** psrc**,** **\***pdst**;**

unsigned int count**;**

count **=** **(**unsigned int**)** **&**Image$$Exec\_RAM1$$Length**;**

psrc **=** **(**unsigned char **\*)&**Load$$Exec\_RAM1$$Base**;**

pdst **=** **(**unsigned char **\*)&**Image$$Exec\_RAM1$$Base**;**

**while** **(**count**--)** **{**

**\***pdst**++** **=** **\***psrc**++;**

**}**

**}**

二.分散加载应用

前面提到过，从NAND Flash启动，对于S3C2410而言，由于片内具有4K的称作"SteppingStone"的SRAM，NAND FLASH的最低4K代码可以自动复制到"SteppingStone"，因此可以将初始化等代码放在NAND FLASH的低4K区域内，其他的代码放置在4K以外，在初始化代码内将这些代码复制到外部SDRAM，从而这些代码可以在外部SDRAM内运行。

1.应用实例描述

先完成初始化操作，并且在初始化代码中将NAND FLASH的4K范围以外的代码（简单起见，这部分代码可以操作LED灯）复制到外部SDRAM中。主要目的是使用分散加载文件以及将NAND FLASH中的数据代码复制到SDRAM中。

2.分散加载文件

（1）将一些初始化代码放在第一个加载区（根区：加载地址和执行地址相同的区域，每一个分散加载描述文件必须至少要有一个根区。），地址范围为：0x0000~0x0fff的4K，其执行区的地址范围也是0x0000~0x0fff的4K，这正好是NAND FLASH启动时自动复制的地址范围。

（2）其他代码放在第2个加载区，从地址0x1000开始，由于这一部分不能自动复制，因此在初始化代码中应该将这一部分复制到外部SDRAM中，其执行区的起始地址为外部SRDAM的地址。

3. 二进制文件烧录

由于有2个加载区，因此生成的二进制文件有2个，文件名对应于相应的执行区名，分别是RAM\_EXEC和SDRAM\_EXEC，需要注意的是，应该将存放初始化代码的加载区对应的二进制文件RAM\_EXEC烧录NAND FLASH的低4K区域，第二个加载区对应的二进制文件SDRAM\_EXEC烧录到4K以后的区域。这个可以通过修改Samsuang的sjf烧录程序实现，原来的烧录程序是按BLOCK（16K）烧录，可以修改为按4K的Section烧录，即将1个Block分为4个Section（4K）。主要修改 k9s1208.c中的K9S1208\_Program函数，需要注意的是，由于NAND FLASH写入前应该擦除，擦除是按Block擦除，由于现在是按Section写，因此应该注意只有在第1次写某一块中的Section前进行擦除，以后再写着一块的其它Section前不能再进行擦除。

这样RAM\_EXEC烧录到0 SECTION，SDRAM\_EXEC烧录到1 SECTION开始的以后的区域中，完成后复位即可。

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LR\_IROM1 0x00000000 0x00080000 { ; 第一个加载域，名字是LR\_IROM1，起始地址0x00000000 大小是0x00080000

ER\_IROM1 0x00000000 0x00080000 { ; 第一个运行时域，名字是ER\_IROM1 起始地址0x00000000 大小事0x00080000

\*.o (RESET, +First) ; IAP第一阶段在FLASH中运行

\*(InRoot$$Sections) ; All library sections that must be in a root region

.ANY (+RO) ; .ANY与\*功能相似，用.ANY可以把已经被指定的具有RW,ZI属性的数据排除

}

RW\_IRAM1 0x10000000 0x00010000 { ; RW data

.ANY(+RW +ZI)

}

RW\_SDRAM1 0xA0000000 0x00800000 { ; RW data

STARTUP\_LPC177X\_8X.o (HEAP) ;HEAP用来定位堆栈的底

\*.LIB(+RW +ZI)

program1.o (+RW, +ZI) //Place all RW and ZI data from program1.o into this exec region

}

}

# 相关格式

* 1. 编号
     1. 1-9）版

1. static无论是全局变量还是局部变量都存储在全局/静态区域，在编译期就为其分配内存，在程序结束时释放，例如：val\_a、val\_d、val\_f、val\_g；
2. const全局变量存储在只读数据段，编译期最初将其保存在符号表中，第一次使用时为其分配内存，在程序结束时释放，例如：val\_c；const局部变量存储在栈中，代码块结束时释放，例如：val\_j；
3. 全局变量存储在全局/静态区域，在编译期为其分配内存，在程序结束时释放，例如：val\_b、val\_e；
   * 1. 1-10）版
4. D能量消耗，降低系统运行成本，并可以延长系统工作时间。近些年许多学者基于操
5. 作
6. 系
7. 统
8. 层
9. 的
10. D
11. P
12. M
13. 技
14. 术对这个问题进行了深入的研究。在本节中，我们
    * 1. （1-10）版
15. 任务之间支持抢占；
16. 备能量消耗，降低系统运行成本，并可以延长系统工作时间。近些年许多学者基于操作系统层的DPM技术对这个问题进行了深入的研究。在本节中，我们首先介
17. 绍
18. 基
19. 于
20. 动
21. 态
22. 调频
23. 备能量
24. 消
25. 系
    * 1. 1. 代码版
26. 低系统运行成本，并可以延长系统工作时间。近些年许多学者基于操作系的的的的统层的DPM技
27. 术
28. 对
29. 这
30. 个
31. 问
32. 题
33. 进
34. 行
35. **typedef** struct \_data\_t **{**
36. int a**;**
37. int b**;**
38. int c**;**
39. **}**data\_t**;**
40. //乱序初始化结构体1
41. data\_t data\_one **=** **{**
42. **.**c **=** 20**,**
43. **.**b **=** 30**,**
44. **.**a **=** 10**,**
45. **};**
46. //乱序初始化结构体2

能量消耗，降低系统运行成本，并

化来降低能耗，利用底层硬件提供的特性，在系统运行时降低系统总体功耗，从而节省大量的能量。

* 1. 图表
     1. 技术

**错误!未找到引用源。**中所示的设备，都是实际中使用的满足本文DPM模型的设备，这些设备的参数来自于文献**错误!未找到引用源。**。

表 2‑1 具有DPM功能的设备的功耗以及*T*be时间

|  |  |  |  |
| --- | --- | --- | --- |
| 设备 | *P*m,i(mW) | *T*m,i(ms) | *T*be(ms) |
| 传感器节点 | 1040/400/270/200/10 | 5/15/20/50 | 8/20/25/50 |
| 网卡（Linksys） | 0.76/0.0 | 2.75 | 3.61 |
| 硬盘（IBM） | 10.2/2.5 | 12.4 | 15.2 |
| Beowolf集群节点 | 1/0.766/0.1/0.01 | 3/7/70 | 6/10/100 |
| 笔记本LCD | 21.1/17.1 | 7.6 | 15.6 |
| 无线网卡 | 0.9/0 | 0.3 | 0.7 |

**错误!未找到引用源。**中的传感器节点具有四种低功耗模式。我们使用该传感器节点的设备参数以

及一些具体任务来描述最理想的DPM策略所需要考虑的综合标准，此外，设备处于睡眠状态所消耗的能量依赖于空闲时间的长度，如图 2‑1所示。



图 2‑1 空闲时间与能量之间凹函数曲线

考虑四个周期性任务组成的任务集*τ* = {*τ*1*, τ*2*, τ*3*, τ*4}，这些任务的释放时间以及执行时间分别为*φ*1=0，*φ*2=100，*φ*3=200，*φ*4=300，*C*1=15，*C*2=78，*C*3=90和*C*4=100，所有任务的周期*T*=400。

本文的章节安排如下：

1. 绪论 介绍本文的研究背景、研究现状以及本文的主要研究内容。

# 参考文献

1. ACPI. 2011. Advanced configuration and power interface standard. http://www.acpi.info