周立功 NXP LPC21xx/22xx 系列 ARM 芯片的启动程序分解

作者：钟常慰　zhongcw1002@126.com

2008.11.25

关于 LPC2200 启动程序分散加载描述文件的叙述

在 ADS LPC2200 的启动模板中有一个 scf 文件夹，其中有 mem_a.scf、mem_b.scf、mem_c.scf 这 3 个文件，这 3 个文件是 ADS 的分散加载机制，其目的是将代码段和数据段分别定位到指定地址上。可以在 Arm Linker 中选择加载路径。

分散装载技术概述：

分散装载技术可以把用户的应用程序分割成多个 RO(只读)运行域和 RW(可读写)运行域(一个存储区域块)，并且给它们制定不同的地址。一个嵌入式系统中，Flash、16 位 RAM、32 位 RAM 都可以存在于系统中，所以，将不同功能的代码定位在特定的位置会大大地提高系统的运行效率。下面是最为常用的 2 种情况：

1、32 位的 RAM 运行速度很快，因此就把中断服务程序作为一个单独的运行域，放在 32 位的 RAM，使它的响应时间达到最快。

2、程序在 RAM 中运行，其效率要远远高于在 ROM 中运行，所以将启动代码(Boot loader)以外的所有代码都复制在 RAM 中运行，可以提高运行效率。

分散装载技术主要完成了 2 个基本的功能：

如何分散。就是如何将输入段组成输出段和域。

如何装载。就是确定装载域和运行域在存储空间里的地址是多少。

域可以分为装载域和运行域

装载域描述运行前输出段和域在 ROM/RAM 里的分布状态，运行域描述了运行时输出段和域在 ROM/RAM 里的分布状态。大多数情况下，映像文件在执行前把它装载到 ROM 里，而当运行时，域里的有些输出段(比如 RW 类型的输出段)必须复制到 RAM 里，程序才能正常运行，所以，在装载和运行时，RW 类的输出段处在不同的位置(地址空间)。

Scatterfile 分散加载文件：

在 scatterfile 中可以为每一个代码或数据区在装载和执行时指定不同的存储区域地址，Scatlertoading 的存储区块可以分成二种类型：

装载区：当系统启动或加载时应用程序的存放区。

执行区：系统启动后，应用程序进行执行和数据访问的存储器区域，系统在实时运行时可以有一个或多个执行块。

映像中所有的代码和数据都有一个装载地址和运行地址(二者可能相同也可能不同，视具体情况而定)。

scatter 文件语法

scatter 文件是一个简单的文本文件，包含一些简单的语法。

```
My Region 0x0000 0x1000  ; 我的名字 My Region 起始地址 0x0000 属性 0x1000
{
;the context of region 这个域的范围
}
```

标题

每个块由一个头标题开始定义，头中至少包含块的名字和起始地址,如(0x0000)，另外还有最大长度等其他一些属性选项(注:这些属性是可选的,如 0x1000)。

内容

块定义的内容包括在紧接的一对花括号内，依赖于具体的系统情况。

1、 一个加载块必须至少含有一个执行块；实践中通常有多个执行块。
2、 一个执行块必须至少含有一个代码或数据段；这些通常来自源文件或库函数等的目标文件；
   通配符号*可以匹配指定属性项中所有没有在文件中定义的余下部分。

有以下几种属性：

RO：只读的代码段和常量

RW：可以读写的全局变量和静态变量

ZI：RW 段中要被初始化为零的变量。

Scatterfile 中的定义要按照系统冲定向后的存储器分布情况进行，在引导程序完成初始化任务后，应该把主程序转移到 RAM 中运行以加快系统的运行速度。

2008.11.25
zhongcw1002@abszcw.co

LPC2200 的分散加载文件分析：

```
ROM_LOAD 0x80000000                        (1)
{
    ROM_EXEC 0x80000000                    (2)
       {   Startup.o (vectors, +First)     (3)
        * (+RO) }                          (4)
    IRAM 0x40000000                        (5)
       {    Startup.o (MyStacks) }         (6)
    STACKS_BOTTOM +0 UNINIT                (7)
       {    Startup.o (StackBottom)   }    (8)
    STACKS 0x40004000 UNINIT               (9)
       {    Startup.o (Stacks)   }         (10)
    ERAM 0x80040000                        (11)
       {    * (+RW,+ZI) }                  (12)
    HEAP +0 UNINIT                         (13)
       {   Startup.o (Heap) }              (14)
    HEAP_BOTTOM 0x80080000 UNINIT          (15)
       {   Startup.o (HeapTop)   }         (16)
}
FLASH_LOAD 0x81000000 0x1000               (17)
{   FLASH_EXEC 0x81000000                  (18)
      {   main.o (+RO) }                   (19)
}
```

(1) 加载时域描述，名称位 ROM_LODA 它的地址为 0x80000000;
    0x80000000 为 LPC 片外 RAM 地址，即将以下的加载的段和域都在 RAM 中。

(2) 第一个运行时域描述。
    ROM_EXEC 描述了执行区的地址，放在第一块定义，其起始地址、空间大小域加载区起始地址、空间大小要一样。

(2)-(4)从起始地址开始放置向量表。Startup.o 是 Startup.s 的目标文件。Vectors 为中断向量表。模块 Startup 位于该加载域的开头(+First),vectors 作为入口点，包含全部的 RO 代码。ARM 在芯片复位之后，系统进入管理模式、ARM 状态，PC(R15)寄存器的值为 0x00000000,所以必须保证用户的向量表代码定位在 0x00000000 处，或者映射到 0x00000000 处(例如向量表代码在 0x80000000 处，通过存储器映射，访问 0x0000000 就是访问（0x80000000)。

(5)-(6)第二运行时域描述。将 MyStacks 堆栈段装载到片内静态 RAM 中。

(7)-(8)将栈底放入堆栈的后面(+0)不进行初始化(UNINIT)，栈底为 Startup 中的 StackBottom。

(9)-(10) 将栈放入地址为 0x40004000 并且不进行初始化(UNINIT)。

(11)-(12 将所有的 RW 和 ZI 段放入外部存储器中以 0x80040000 为开头的地址中。并且全部清零(+ZI)外部 RAM 中指定的区域。

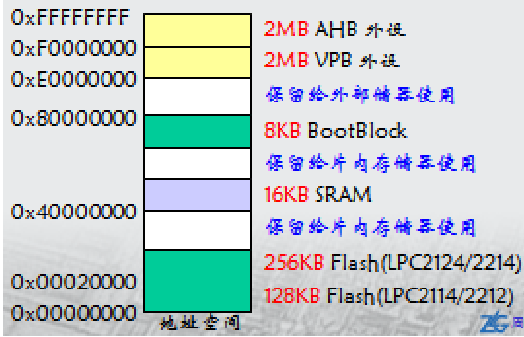
(13)-(14)在 RW ZI 段后放入堆底(Startup.o(Heap))并且不进行初始化。

(15)-(16)将堆定放入外部 RAM 中(0x80080000)。

(17)-(19)自己添加的加载代码，把 main.c 的目标文件加载到片外 Flash 中并且占用了 0x1000 的大小。

2008.11.25
zhongcw1002@abszcw.cc

```
;********************************************************************************************/
;** File Name: men_a.scf
ROM_LOAD 0x80000000                          // ROM加载
{
    ROM_EXEC 0x80000000                      // ROM执行（起始地址）程序0x80000000   外部存储区
        {   Startup.o (vectors, +First)      // Startup.o文件（向量，程序入口）
            * (+RO)}                         // 只读
    IRAM 0x40000000                          // Indexed Random Access Method,索引随机存取方法0x40000000
        {   Startup.o (MyStacks)}            // Startup（堆栈）装载到片内SRAM中
    STACKS_BOTTOM +0 UNINIT                  // 堆栈栈底，将栈底放入堆栈的后面(+0)，不初始化
        {   Startup.o (StackBottom)}         // 将栈放入地址为0x40004000 并且不进行初始化(UNINIT)
    STACKS 0x40004000 UNINIT                 // 堆栈0x40004000          16kb的SRAM片内存储区
        {   Startup.o (Stacks)}              // Startup.o文件（堆栈）
    ERAM 0x81000000                          // 将所有的RW和ZI段放入外部存储器中0x81000000为开头的地址中
        {   * (+RW,+ZI)}                     // 并且全部清零(+ZI)外部RAM中指定的区域读写（无输入ZI）
    HEAP +0 UNINIT                           // 在RW ZI段后放入堆底(Startup.o(Heap))并且不进行初始化
        {   Startup.o (Heap)}                // Startup.o文件（堆栈）
    HEAP_BOTTOM 0x81080000 UNINIT            // 将堆定放入外部RAM中0x81080000
        {   Startup.o (HeapTop)}             // Startup.o文件（堆栈栈顶）
}
;********************************************************************************************/
;** File Name: men_b.scf
ROM_LOAD 0x80000000
{
    ROM_EXEC 0x80000000
        {   Startup.o (vectors, +First)
            * (+RO)}
    IRAM 0x40000000
        {   Startup.o (MyStacks)}
    STACKS_BOTTOM +0 UNINIT
        {   Startup.o (StackBottom)}
    STACKS 0x40004000 UNINIT
        {   Startup.o (Stacks)}
    ERAM 0x80040000
        {   * (+RW,+ZI)}
    HEAP +0 UNINIT
        {   Startup.o (Heap)}
    HEAP_BOTTOM 0x80080000 UNINIT
        {   Startup.o (HeapTop)}
}
```




```
;********************************************************************************************/
;** File Name: men_c.scf
ROM_LOAD 0x0
{
    ROM_EXEC 0x00000000
        {   Startup.o (vectors, +First)
            * (+RO)}
    IRAM 0x40000000
        {   Startup.o (MyStacks)}
    STACKS_BOTTOM +0 UNINIT
        {  Startup.o (StackBottom)}
    STACKS 0x40004000 UNINIT
        {  Startup.o (Stacks)}
    ERAM 0x80000000
        {  * (+RW,+ZI)}
    HEAP +0 UNINIT
        {  Startup.o (Heap)}
    HEAP_BOTTOM 0x80080000 UNINIT
        {  Startup.o (HeapTop)}
}
;/********************************************************************************************
```

加载文件的更多分解说明，网上了解资料。只作参考

有如下分散加载文件：

```
ROM_LOAD0x00000000                 // 程序起始点（程序在Flash中） Origination Point of Code (Code in Flash)
{
    ROM_EXEC 0x00000000            // 起始程序执行 Origination Point of Executing
      { Startup.o (vectors, +First)
       * (+RO) }
    IRAM 0x40000040                // 内部SRAM起始点 Origination Point of Internal SRAM
      { Startup.o (MyStacks) }     // 0x40000000 ~0x4000003F for Vector
      STACKS_BOTTOM +0 UNINIT
      { Startup.o (StackBottom) }
      STACKS 0x40004000 UNINIT     // End Point of Internal SRAM
      { Startup.o (Stacks) }
    ERAM 0x81000000                // 外部RAM起始点 Origination Point of External SRAM
      { * (+RW,+ZI) }
      HEAP +0 UNINIT
      { Startup.o (Heap) }
      HEAP_BOTTOM 0x81800000 UNINIT // 外部RAM结束点 End Point of External SRAM
      { Startup.o (HeapTop) }
}
```

其中， ROM_LOAD为加载区的名称，其后面的0x00000000表示加载区的起始地址（存放程序代码的起始地址），也可以在后面添加其空间大小，如“ROM_LOAD 0x00000000 0x20000”表示加载区起始地址为0x00000000，大小为128K字节；ROM_EXEC描述了执行区的地址，放在第一块位置定义，其起始地址、空间大小与加载区起始地址、空间大小要一致。从起始地址开始放置向量表（即Startup.o（vectors，+First），其中Startup.o为Startup.s的目标文件），接着放置其他代码（即映像文件）（即*（RO））；变量区IRAM的起始地址为0x4000000040，放置Startup.o（MyStacks）；变量区ERAM的起始地址为0x80000000，放置出Startup.o文件之外的其他文件的变量（即*（+RW，+ZI））；紧靠ERAM变量区之后的是系统堆空间（HEAP），放置描述为Startup.o（Heap）；堆栈区STACKS使用片内RAM，由于ARM的堆栈一般采用满递减堆栈，所以堆栈区的起始地址设置为0x40004000，放置描述为Startup.o（Stacks）

2. 使用地址不连续的内存（LPC2368）

Lpc2368一共有56K的RAM，其中通用Ram32K，地址为0x40000000~0x40007fff；8KB的USB专用RAM，地址0x7fd00000~0x7fd01ffff；16KB Ethernet专用RAM，地址为0x7fe00000~0x7fe03fff；以上的USB和Ethernet专用RAM也可用做通用RAM，需要做如下设置：（1）target.c中将USB和Ethernet功能打开，需要设置PCONP寄存器，详见Datasheet。（2）设置分散加载文件，分配这两段内存。

在DebugInRam模式下，有如下分散加载文件：

```
ROM_LOAD 0x40000000
{
   ROM_EXEC 0x40000000 //加载映像文件（通用RAM首地址）
      { Startup.o (vectors, +First)
     * (+RO) }
   IRAM 0x40007000 //用户堆栈
      { Startup.o (MyStacks) }
      STACKS 0x40008000 UNINIT //系统堆栈
      { Startup.o (Stacks) }
   ERAM 0x7fe00000 /*变量，放置与Ethernet专用RAM首地址*/
      { * (+RW,+ZI) }
      HEAP +0 UNINIT
      { Startup.o (Heap) }
}
```

3．分散使用Flash地址（LPC2368）

项目中，要求将片内Flash起始几个扇区空出来留作他用，或者当用到的Flash地址不连续的时候，都可用以下方法来编写分散加载文件：

```
ROM_LOAD 0x00000000
{
   ROM_EXEC0x00000000   /*中断向量表*/
     { Startup.o (vectors,+First) }
   ……
   ROM_LOAD1 0x00004000 //加载映像文件，从第四个扇区开始
     { ROM_EXEC10x00004000
     {
       * (+RO)
      }
}
```

值得注意的是，中断向量表必须放在flash起始地址处，否则无法启动。根据以上分散加载文件编译生成的Hex文件会有两个，分别如下：

```
Hex1：
:020000040000FA
:1000000018F09FE518F09FE518F09FE518F09FE5C0
……
Hex2：
:020000040000FA
:1040000090808FE20F0098E8080080E0081081E0BF
。。。
```

可以看出，生成的两段Hex文件的起始地址是不同的，其中一段为中断向量表；另一段为用户映像文件。

4．固定变量内存地址

嵌入式开发中，有时会需要在同一片内的不同段程序（比如Bootloader和主程序间）间传递数据，这时候往往需要固定变量地址。一般来言，C语言编写的程序，变量地址是由C编译器来分配内存的，程序员无法实现知道变量地址。而ADS中的分散加载文件可以告知编译器，固定某些变量的地址，如下：

```
ROM_LOAD 0x00000000
{
    ROM_EXEC 0x00000000
      { Startup.o (vectors, +First)
      * (+RO) }
    RAM 0x40000000 UNINIT //Mfile.c中的所有变量地址从0x40000000开始
      { Mfile.O(+RW,+ZI) }
    IRAM 0x40000010
      { Startup.o (MyStacks)
      * (+RW,+ZI) }
      HEAP +0 UNINIT
      { Startup.o (Heap) }
    STACKS 0x40004000 UNINIT
      { Startup.o (Stacks) }
}
```

上述分散加载文件固定了Mfile.c中变量的起始地址，以这种方法，可以固定任何全局变量的地址，以便其被其他系统访问。

```
;/********************************************************************************
;** File name:              Startup.s        // 起始文件
;定义堆栈的大小
SVC_STACK_LEGTH         EQU         0         // 管理堆栈长度
FIQ_STACK_LEGTH         EQU         0         // 快速中断堆栈长度
IRQ_STACK_LEGTH         EQU         256       // 通用中断堆栈长度
ABT_STACK_LEGTH         EQU         0         // 中止堆栈长度
UND_STACK_LEGTH         EQU         0         // 未定义堆栈长度

NoInt       EQU 0x80

USR32Mode   EQU 0x10                          // 用户模式
SVC32Mode   EQU 0x13                          // 管理模式
SYS32Mode   EQU 0x1f                          // 系统模式
IRQ32Mode   EQU 0x12                          // 通用中断模式
FIQ32Mode   EQU 0x11                          // 快速中断模式
                                              // 以下几个定义是确定寄存器所在地址（保证前期初始化作业）书本191-193页
PINSEL2     EQU 0xE002C014                    // 定义引脚功能选择寄存器2在芯片中所在的地址（0xE002C014）（不管内容）

BCFG0       EQU 0xFFE00000                    // 存储器组0所在芯片位置（0xFFE00000）设定与外设的读写及间隔速度设定
BCFG1       EQU 0xFFE00004                    // 存储器组1所在芯片位置（0xFFE00004）设定与外设的读写及间隔速度设定
BCFG2       EQU 0xFFE00008                    // 存储器组2所在芯片位置（0xFFE00008）设定与外设的读写及间隔速度设定
BCFG3       EQU 0xFFE0000C                    // 存储器组3所在芯片位置（0xFFE0000C）设定与外设的读写及间隔速度设定

    IMPORT __use_no_semihosting_swi
    IMPORT __use_two_region_memory

;引入的外部标号在这声明
    IMPORT  FIQ_Exception                     // 快速中断异常处理程序
    IMPORT  __main                            // C语言主程序入口
    IMPORT  TargetResetInit                   // 目标板基本初始化

;给外部使用的标号在这声明                       // 这些标号都是代表一段字节的首地址，可参见后面的部分设定
    EXPORT  bottom_of_heap                    // heap的底部
    EXPORT  bottom_of_Stacks                  // stack的底部
    EXPORT  top_of_heap                       // heap的顶部
    EXPORT  StackUsr

    EXPORT  Reset                             // 复位
    EXPORT __user_initial_stackheap           // 库函数初始化堆和栈

    CODE32                                    // 32位ARM代码

    AREA    vectors,CODE,READONLY             // 只读程序入口
        ENTRY                                 // 进入
;中断向量表
Reset
        LDR     PC, ResetAddr                // 复位起始地址
        LDR     PC, UndefinedAddr            // 未定义异常向量地址
        LDR     PC, SWI_Addr                 // 软件复位向量地址
        LDR     PC, PrefetchAddr             // 预指令中止
        LDR     PC, DataAbortAddr            // 数据异常向量地址
        DCD     0xb9205f80                   // 将所有向量表合计为0的一个指定数据
        LDR     PC, [PC, #-0xff0]            // IRQ（ 该指令会读取VICVectAddr寄存器的值,然后放入PC指针）
        LDR     PC, FIQ_Addr                 // 快速中断地址
;给每一个向量分配连续的字存储单元
ResetAddr           DCD     ResetInit            // 将ResetInit入口地址赋予ResetAddr         复位初始化
UndefinedAddr       DCD     Undefined            // 将Undefined入口地址赋予UndefinedAddr     未定义
SWI_Addr            DCD     SoftwareInterrupt    // 将SoftwareInterrupt入口地址赋予SWI_Addr 软件复位
PrefetchAddr        DCD     PrefetchAbort        // 将PrefetchAbort入口地址赋予PrefetchAddr 预指令中止
DataAbortAddr       DCD     DataAbort            // 将DataAbort入口地址赋予DataAbortAddr     数据异常
```

```
Nouse               DCD     0                    // 将Nouse赋予0
IRQ_Addr            DCD     0                    // 将IRQ_Addr赋予0                   通用中断
FIQ_Addr            DCD     FIQ_Handler          // 将FIQ_Handler入口地址赋予FIQ_Addr  快速中断

;未定义指令
Undefined
        B       Undefined           // 原地循环

;软中断
SoftwareInterrupt
        B       SoftwareInterrupt   // 原地循环

;取指令中止
PrefetchAbort
        B       PrefetchAbort       // 原地循环

; 取数据中止
DataAbort
        B       DataAbort           // 原地循环

;快速中断
FIQ_Handler
        STMFD   SP!, {R0-R3, LR}     // 入栈
        BL      FIQ_Exception        // 调快速中断程序
        LDMFD   SP!, {R0-R3, LR}     // 出栈
        SUBS    PC,  LR,  #4         // 返回当初的地址

;/********************************************************************************************
;** unction name   函数名称:     InitStack
;** Descriptions   功能描述:     Initialize the stacks  初始化堆栈
InitStack
        MOV     R0, LR               // 暂存链接地址
;设置管理模式堆栈
        MSR     CPSR_c, #0xd3        // 设为管理状态（1101 0011），IRQ、FIQ为允许状态
        LDR     SP, StackSvc         // 装入管理模式的堆栈
;设置中断模式堆栈
        MSR     CPSR_c, #0xd2        // 设为中断状态（1101 0011），IRQ、FIQ为允许状态
        LDR     SP, StackIrq         // 装入管理模式的堆栈
;设置快速中断模式堆栈
        MSR     CPSR_c, #0xd1        // 设为快速中断状态（1101 0011），IRQ、FIQ为允许状态
        LDR     SP, StackFiq         // 装入快速中断模式的堆栈
;设置中止模式堆栈
        MSR     CPSR_c, #0xd7        // 设为中止状态（1101 0011），IRQ、FIQ为允许状态
        LDR     SP, StackAbt         // 装入中止模式的堆栈
;设置未定义模式堆栈
        MSR     CPSR_c, #0xdb        // 设为未定义状态（1101 1100），IRQ、FIQ为允许状态
        LDR     SP, StackUnd         // 装入未定义模式的堆栈
;设置系统模式堆栈
        MSR     CPSR_c, #0xdf        // 设为系统状态（1101 1111），IRQ、FIQ为允许状态
        LDR     SP, =StackUsr        // 装入系统模式的堆栈

        MOV     PC, R0               // 返回链接地址
;/********************************************************************************************
;** unction name   函数名称:     ResetInit
;** Descriptions   功能描述:     RESET  复位入口
ResetInit
;初始化外部总线控制器，根据目标板决定配置
        LDR     R0, =PINSEL2
    IF :DEF: EN_CRP                  // 判断是否已经定义了EN_CRP
        LDR     R1, =0x0f814910      // 是的，已经定义了，第三位使用为P1.31:26用着GPIO
    ELSE
```

```
        LDR     R1, =0x0f814914         // 否则，第三位告诉P1.31:26用作为调试端口
    ENDIF
        STR     R1, [R0]                // 设定好管脚选择寄存器2（PINSEL2）

        LDR     R0, =BCFG0              // 设定好读写外设的延时，课本187页
        LDR     R1, =0x1000ffef
        STR     R1, [R0]
                                        // 简单说明：BANK0:8000 0000 – 80FF FFFF 配置寄存器BCFG0
        LDR     R0, =BCFG1              //          ：BANK1:8100 0000 – 81FF FFFF 配置寄存器BCFG1
        LDR     R1, =0x1000ffef         //          ：BANK2:8200 0000 – 82FF FFFF 配置寄存器BCFG2
        STR     R1, [R0]                //          ：BANK3:8300 0000 – 83FF FFFF 配置寄存器BCFG3
                                        // 根据自己的配置的BANK0和BANK1来确定，在本板块中只是有了BANK0和BANK1
;        LDR     R0, =BCFG2             // BANK2在本例中没有使用，如果配置了也需要设定外设读写速度
;        LDR     R1, =0x2000ffef
;        STR     R1, [R0]

;        LDR     R0, =BCFG3             // BANK3在本例中没有使用，如果配置了也需要设定外设读写速度
;        LDR     R1, =0x2000ffef
;        STR     R1, [R0]

        BL      InitStack               // 初始化堆栈
        BL      TargetResetInit         // 目标板基本初始化
                                        // 跳转到c语言入口
        B       __main
;/*********************************************************************************************
;** unction name    函数名称:     __user_initial_stackheap
;** Descriptions    功能描述:     库函数初始化堆和栈，不能删除
;** input parameters    输  入:   reference by function library 参考库函数手册
;** Returned value      输  出:   reference by function library 参考库函数手册
__user_initial_stackheap
    LDR     r0,=bottom_of_heap      // 将bottom_of_heap空间对应的地址赋予r0
;   LDR     r1,=StackUsr            // 将StackUsr空间对应的地址赋予r1
    LDR     r2,=top_of_heap         // 将top_of_heap空间对应的地址赋予r2
    LDR     r3,=bottom_of_Stacks    // 将bottom_of_Stacks空间对应的地址赋予r3
    MOV     pc,lr                   // 返回链接地址

StackSvc        DCD     SvcStackSpace + (SVC_STACK_LEGTH - 1)* 4    // StackSvc赋值
StackIrq        DCD     IrqStackSpace + (IRQ_STACK_LEGTH - 1)* 4    // StackIrq赋值
StackFiq        DCD     FiqStackSpace + (FIQ_STACK_LEGTH - 1)* 4    // StackFiq赋值
StackAbt        DCD     AbtStackSpace + (ABT_STACK_LEGTH - 1)* 4    // StackAbt赋值
StackUnd        DCD     UndtStackSpace + (UND_STACK_LEGTH - 1)* 4   // StackUnd赋值

;/*********************************************************************************************
;** unction name    函数名称:     CrpData
;** Descriptions    功能描述:     encrypt the chip    加密芯片
```

局部解释：

"."表示当前代码地址； INFO是个伪指令，用于输出出错信息。

整个程序做的事情用一句话来概括，就是保证内部FLASH的0x1fc处为0x87654321。这样，芯片在下次复位时就会加密。

```
  IF :DEF: EN_CRP        //声明一个ASM编译条件
    IF . >= 0x1fc
    INFO  1,"\nThe data at 0x000001fc must be 0x87654321.\nPlease delete some source before this line."
    ENDIF"
```

用于检查当前地址是不是已经过了0x1fc.

```
"CrpData
  WHILE . < 0x1fc
  NOP
  WEND
CrpData1
  DCD   0x87654321      ; 当此数为0x87654321时，用户程序被保护 */就是将0x87654321赋值到0x1fc地址处
  ENDIF"
```

是在用NOP指令填充0x1fc之前的地址，等填到0x1fc时，就填入0x87654321

```
IF :DEF: EN_CRP                 // 判断是否已经定义了EN_CRP
        IF  . >= 0x1fc          // 判断当前地址是否超过了0x0000 001FC
        INFO    1,"\nThe data at 0x000001fc must be 0x87654321.\nPlease delete some source before this line."
        ENDIF
CrpData
    WHILE . < 0x1fc             //判断还没有超过0x0000 001FC
    NOP                         // 用NOP指令填充，直到0x0000 001FC为止
    WEND                        // 是0x0000 001FC跳出
CrpData1
    DCD     0x87654321          // 当到达0x0000 001FC时，将x87654321保存进去时，用户程序被（加密）保护
    ENDIF

;/* 分配堆栈空间*/
        AREA    MyStacks, DATA, NOINIT, ALIGN=2 // 分配一个名为MyStacks的数据空间，无需初始化，地址对齐为半字
SvcStackSpace      SPACE   SVC_STACK_LEGTH * 4  // 管理模式堆栈空间     = 管理堆栈长度*4（字节）
IrqStackSpace      SPACE   IRQ_STACK_LEGTH * 4  // 中断模式堆栈空间     = 中断堆栈长度*4（字节）
FiqStackSpace      SPACE   FIQ_STACK_LEGTH * 4  // 快速中断模式堆栈空间 = 快速中断堆栈长度*4（字节）
AbtStackSpace      SPACE   ABT_STACK_LEGTH * 4  // 中止模式堆栈空间     = 中止堆栈长度*4（字节）
UndtStackSpace     SPACE   UND_STACK_LEGTH * 4  // 未定义模式堆栈空间   = 未定义堆栈长度*4（字节）

        AREA    Heap, DATA, NOINIT              // 分配一个名为Heap的数据堆栈，无需初始化
bottom_of_heap    SPACE   1                     // 设定一个名字为bottom_of_heap的空间，占用一个字
        AREA    StackBottom, DATA, NOINIT       // 分配一个名为StackBottom的数据堆栈，无需初始化
bottom_of_Stacks    SPACE   1                   // 设定一个名字为bottom_of_Stacks的空间，占用一个字
        AREA    HeapTop, DATA, NOINIT           // 分配一个名为HeapTop的数据堆栈，无需初始化
top_of_heap                                     // 分配一段名为top_of_heap的未知长度的空间
        AREA    Stacks, DATA, NOINIT            // 分配一个名为HeapTop的数据堆栈，无需初始化
StackUsr                                        // 分配一段名为StackUsr的未知长度的空间

    END
;/*********************************************************************************************************
```

```
;/*********************************************************************************************
;** File Name: IRQ.s
NoInt       EQU 0x80

USR32Mode   EQU 0x10                            // 用户模式
SVC32Mode   EQU 0x13                            // 管理模式
SYS32Mode   EQU 0x1f                            // 系统模式
IRQ32Mode   EQU 0x12                            // 通用中断模式
FIQ32Mode   EQU 0x11                            // 快速中断模式

    CODE32                                      // 32位ARM程序
    AREA    IRQ,CODE,READONLY
    MACRO
$IRQ_Label HANDLER $IRQ_Exception_Function     // 用IRQ_Label来替代IRQ_Exception_Function，汇编调用C函数？？

        EXPORT  $IRQ_Label                      // 输出的标号
        IMPORT  $IRQ_Exception_Function         // 引用的外部标号

$IRQ_Label
        SUB     LR, LR, #4                      // 计算返回地址
        STMFD   SP!, {R0-R3, R12, LR}           // 保存任务环境
        MRS     R3, SPSR                        // 保存状态
        STMFD   SP, {R3,LR}^                    // 保存SPSR和用户状态的SP,注意不能回写
                                                // 如果回写的是用户的SP，所以后面要调整SP
        NOP
        SUB     SP, SP, #4*2

        MSR     CPSR_c, #(NoInt | SYS32Mode)    // 切换到系统模式

        BL      $IRQ_Exception_Function         // 调用c语言的中断处理程序

        MSR     CPSR_c, #(NoInt | IRQ32Mode)    // 切换回irq模式
        LDMFD   SP, {R3,LR}^                    // 恢复SPSR和用户状态的SP,注意不能回写
                                                // 如果回写的是用户的SP，所以后面要调整SP
        MSR     SPSR_cxsf, R3
        ADD     SP, SP, #4*2                     //

        LDMFD   SP!, {R0-R3, R12, PC}^           // 全部入栈
    MEND

;/* 以下添加中断句柄，用户根据实际情况改变*/
;Timer0_Handler  HANDLER Timer0

    END
;/*********************************************************************************************
```

来资一段网上解释：

$IRQ_Label HANDLER $IRQ_Exception_Function

HANDLER是宏名，有个参数IRQ_Exception_Function，$代表参数会被替换,IRQ_Label是标号。。。

例：$SPI_Exception HANDLER $IRQ_Spi

SPI_Exception 替代 IRQ_Label

IRQ_Label 替代 IRQ_Exception_Function

变为：

```
 SPI_Exception
    SUB   LR, LR, #4                  ; 计算返回地址
    STMFD SP!, {R0-R3, R12, LR}       ; 保存任务环境
    MRS   R3, SPSR                    ; 保存状态
    STMFD  SP, {R3, SP, LR}^          ; 保存用户状态的R3,SP,LR,注意不能回写
                                      ; 如果回写的是用户的SP，所以后面要调整SP
 BL    IRQ_Spi ; 调用c语言的中断处理程序
```

（来自网上的一段解释）详细内容请参见清远见嵌入式培训专家《ARM系列处理器应用技术完全手册》第13章 15页

获取映像符号

1 获得连接器预定义符号

连接器定义了一些包含$$的符号。这些符号及其他所有包含$$的名称都是 ARM 的保留字。这些符号被用于指定域的基地址，输出段的基地址和输入段的基地址及其大小。

你可以在你的汇编语言程序中引用这些符号地址，把它们用作可重定位的地址，也可能在 C 或 C++代码中使用 exern 关键字来引用它们。

1．1 与域相关的符号

当 armlink 生成映像时产生与域相关的符号。对每个包含一个(被 0 初始化)ZI 输出段执行域来说，armlink 都产生包含了$$ZI$$的附加符号。

Load$$region_name$$Base	域的装载地址
Image$$region_name$$Base	域的执行地址
Image$$region_name$$Length	执行域的长度（4*字节）
Image$$region_name$$Limit	超出执行域结尾的字节地址
Image$$region_name$$ZI$$Base	在此域中 ZI 输出段的执行地址
Image$$region_name$$ZI$$Length	ZI 输出段的长度（4*字节）
Image$$region_name$$ZI$$Limit	超出执行域中 ZI 输出段结尾的字节地址

在 ZI 域以上放置堆栈：stack 和 heap

通常使用与域相关的符号来在 ZI 域以上直接设置堆栈。请参考 ADS Development Giude 中有关 ROM 的章节。

1．2 段相关的符号

一个简单的映像有三个输出段（RO，RW 和 ZI），这三个段产生三个执行域。对每个映像中的输入段，armlink 都产生如下的输入符号：

Image$$RO$$Base	RO 输出段的起始地址
Image$$RO$$Limit	超出 RO 输出段结尾的第一个字节地址
Image$$RW$$Base	RW 输出段的起始地址
Image$$RW$$Limt	超出 RW 输出段结尾的第一个字节地址
Image$$ZI$$Base	ZI 输出段的起始地址

ARM连接器生成的符号

又发现一个不明白的地方：

|Image$$RO$$Limit|

|Image$$RW$$Base|

|Image$$ZI$$Base|

|Image$$ZI$$Limit|

这四个变量表示意思倒是能在注释中看到，但是找来找去，还愣是没有在一大堆文件中找到什么时候定义过这几个变量，最后不得不找Baidu了，结果如下：

ARM 连接器定义了一些包含$$的符号。这些符号及其他所有包含$$的名称都是 ARM 的保留字。这些符号被用于指定域的基地址，输出段的基地址和输入段的基地址及其大小。

我们可以自己的汇编语言程序中引用这些符号地址，把它们用作可重定位的地址，也可能在 C 或 C++代码中使用 exern 关键字来引用它们。

因为$在 ARM 汇编中有特殊的含义，所以在使用 Image$$RO$$Base 这样的符号时需要在两个竖线“|”之间来告诉汇编器这里的$$不用处理

ARM汇编器的内置变量

内置变量的设置不能用 SETA,SETL 或 SETS 等指示符来设置,只能用表达式或条件来设置.

例如:
IF {ARCHITECTURE} = "4T"

内置变量	变量含义
{PC}或.	当前指令的地址
{VAR}或@	存储区位置计数器的当前值
{TRUE}	逻辑常量真
{FALSE}	逻辑常量假
{OPT}	当前设置列表选项值,OPT 用来保存当前列表选项,改变选项值,恢复它的原始值
{CONFIG}	如果汇编器汇编 ARM 代码,则值为 32;如果汇编器汇编 Thumb 代码,则值为 16
{ENDIAN}	如果汇编器在大端模式下,则值为 big;如果汇编器在小端模式下,则值为 little
{CODESIZE}	如果汇编器汇编 ARM 代码,则值为 32;如果汇编器汇编 Thumb 代码,则值为 16,与{CONFIG}同义
{CPU}	选定的CPU名,缺省时为ARM7TDMI
{FPU}	选定的FPU名,缺省时为SoftVFP
{ARCHITECTURE}	选定的ARM体系结构的值;3,3M,4,4T和4TxM
{PCSTOREOFFSET}	STR pc,[…]或 STMRb,[…PC]指令的地址和 PC 存储值之间的偏移量
{ARMASM_VERSION}或| ads $ version |	ARM 汇编器的版本号,为整数

关于 ARM 汇编里的特殊符号

先前企图全部靠自己写一个 bootloader，结果尝试了下，花了 4 天时间查各种技术资料，写了个 startup.s 文件出来，写的过程中才发现，原来还有很多问题是我基本上不知道的，比如说如何进行 ARM 的位操作、如何将堆栈设置到 RAM 中、UART 的波特率计算方法等问题。

在边写边查资料的过程中，我又发现了别人的一些程序我看不懂……因为除了 EQU、DCD 等我基本不用伪指令……
所以我开始看 44B0 BootLoader 的范例程序，可能是人家水平实在比较高，也可能是俺的水平确实有限，总之是有些地方看不怎么懂，特别是一些个特殊符号，现特将那些个麻烦的符号总结下：

符号	对应指令	含义	示例
^	MAP	定义结构化内存表	MAP 4096;内存表首地址为4096
#	FIELD	定义内存表中的数据，结合MAP指使用	STACKSVC FIELD 256; 定义从4096开始的256字节为SVC的堆栈空间
%	SPACE	分配一块内存，并用“0”初始化	DataStruc SPACE 280分配280字节内存并初始化

来自一个网站的解释：

对于刚学习 ARM 的人来说，如果分析它的启动代码，往往不明白下面几个变量的含义：

|Image$$RO$$Limit|

|Image$$RW$$Base|

|Image$$ZI$$Base|。

首先申明我使用的调试软件为 ADS1.2，当我们把程序编写好以后，就要进行编译和链接了，在 ADS1.2 中选择 MAKE 按钮，会出现一个 Errors and Warnings 的对话框，在该栏中显示编译和链接的结果，如果没有错误，在文件的最后应该能看到 Image component sizes，后面紧跟的依次是 Code，RO Data ，RW Data ，ZI Data ，Debug 各个项目的字节数，最后会有他们的一个统计数据：

Code 163632 ，RO Data 20939 ，RW Data 53 ，ZI Data 17028

Tatal RO size (Code+ RO Data)　　184571 (180.25kB)

Tatal RW size(RW Data+ ZI Data)　　17081(16.68 kB)

Tatal ROM size(Code+ RO Data+ RW Data)　　184624(180.30 kB)

后面的字节数是根据用户不同的程序而来的，下面就以上面的数据为例来介绍那几个变量的计算。

在 ADS 的 Debug Settings 中有一栏是 Linker/ARM Linker，在 output 选项中有一个 RO base 选项，下面应该有一个地址，我这里是 0x0c100000，后面的 RW base 地址是 0x0c200000，然后在 Options 选项中有 Image entry point ，是一个初始程序的入口地址，我这里是 0x0c100000 。

有了上面这些信息我们就可以完全知道这几个变量是怎么来的了：

|Image$$RO$$Base| = Image entry point = 0x0c100000 ；表示程序代码存放的起始地址

|Image$$RO$$Limit| = 程序代码起始地址+代码长度+1 = 0x0c100000+Tatal RO size+1

= 0x0c100000 + 184571 + 1　　= 0x0c100000 +0x2D0FB + 1

= 0x0c12d0fc

|Image$$RW$$Base| = 0x0c200000 ；由 RW base 地址指定

|Image$$RW$$Limit| =|Image$$RW$$Base| + RW Data 53

= 0x0c200000 + 0x37（4 的倍数,0 到 55，共 56 个单元）

= 0x0c200037

|Image$$ZI$$Base| = |Image$$RW$$Limit| + 1 = 0x0c200038

|Image$$ZI$$Limit| = |Image$$ZI$$Base| + ZI Data 17028

=0x0c200038 + 0x4284

=0x0c2042bc

也可以由此计算：

|Image$$ZI$$Limit| = |Image$$RW$$Base| +TatalRWsize(RWData+ZIData) 17081（17028 + 53）

=0x0c200000+0x42b9+3（要满足 4 的倍数）

=0x0c2042bc

```
/****************************************Copyright (c)**************************************************
** File name:              target.h
#ifndef __TARGET_H
#define __TARGET_H
    #ifdef __cplusplus             //如果这是一段cpp的代码，那么加入extern "C"{和}处理其中的代码
      extern "C" {
    #endif
    #ifndef IN_TARGET
      extern void Reset(void);     //复位
      extern void TargetInit(void);  //目标板初始化
    #endif
    #ifdef __cplusplus
    }
    #endif
 #endif
*******************************************************************************************************/
```

时常在 cpp 的代码之中看到这样的代码:（摘自网上）

```
1 #ifdef __cplusplus
2  extern "C" {        // 为什么括号中先有个#endif，最后又有#ifdef __cplusplus???
3   #endif             // 第 1 行和第 3 行对应，第 5 行和第 7 行对应

4   //一段代码          // {}表示这个括号范围内的都和 c 的函数兼容，没有括号的话，extern c 只修饰后面的一个句子

5   #ifdef __cplusplus
6 }
7 #endif
```

这样的代码到底是什么意思呢？首先，__cplusplus 是 cpp 中的自定义宏，那么定义了这个宏的话表示这是一段 cpp 的代码，也就是说，上面的代码的含义是：如果这是一段 cpp 的代码，那么加入 extern "C"{和}处理其中的代码。

要明白为何使用 extern "C"，还得从 cpp 中对函数的重载处理开始说起。在 c++中，为了支持重载机制，在编译生成的汇编码中，要对函数的名字进行一些处理，加入比如函数的返回类型等等.而在 C 中，只是简单的函数名字而已，不会加入其他的信息.也就是说:C++和 C 对产生的函数名字的处理是不一样的.

比如下面的一段简单的函数，我们看看加入和不加入 extern "C"产生的汇编代码都有哪些变化:

```
int f(void)
{
return 1;
}
```

在加入 extern "C"的时候产生的汇编代码是:

```
.file "test.cxx"
.text
```

```
.align 2
.globl _f
.def _f; .scl 2; .type 32; .endef
_f:
pushl %ebp
movl %esp， %ebp
movl $1， %eax
popl %ebp
ret
```

但是不加入了 extern "C"之后

```
.file "test.cxx"
.text
.align 2
.globl __Z1fv
.def __Z1fv; .scl 2; .type 32; .endef
__Z1fv:
pushl %ebp
movl %esp， %ebp
movl $1， %eax
popl %ebp
ret
```

两段汇编代码同样都是使用 gcc -S 命令产生的，所有的地方都是一样的，唯独是产生的函数名，一个是_f，一个是__Z1fv。

明白了加入与不加入 extern "C"之后对函数名称产生的影响，我们继续我们的讨论:为什么需要使用 extern "C"呢？C++之父在设计 C++之时，考虑到当时已经存在了大量的 C 代码，为了支持原来的 C 代码和已经写好 C 库，需要在 C++中尽可能的支持 C，而 extern "C"就是其中的一个策略。

试想这样的情况:一个库文件已经用 C 写好了而且运行得很良好，这个时候我们需要使用这个库文件，但是我们需要使用 C++来写这个新的代码。如果这个代码使用的是 C++的方式链接这个 C 库文件的话，那么就会出现链接错误.我们来看一段代码:首先，我们使用 C 的处理方式来写一个函数，也就是说假设这个函数当时是用 C 写成的:

```
//f1.c
extern "C"
{
void f1()
{
return;
}
}
```

编译命令是:gcc -c f1.c -o f1.o 产生了一个叫 f1.o 的库文件。再写一段代码调用这个 f1 函数:

```
// test.cxx
//这个 extern 表示 f1 函数在别的地方定义，这样可以通过
//编译，但是链接的时候还是需要
//链接上原来的库文件.
extern void f1();

int main()
{
f1();

return 0;
}
```

通过 gcc -c test.cxx -o test.o 产生一个叫 test.o 的文件。然后，我们使用 gcc test.o f1.o 来链接两个文件，可是出错了，错误的提示是:

test.o(.text + 0x1f):test.cxx: undefine reference to **'f1()'**

也就是说，在编译 test.cxx 的时候编译器是使用 C++的方式来处理 f1()函数的，但是实际上链接的库文件却是用 C 的方式来处理函数的，所以就会出现链接过不去的错误:因为链接器找不到函数。

因此，为了在 C++代码中调用用 C 写成的库文件，就需要用 extern "C"来告诉编译器:这是一个用 C 写成的库文件，请用 C 的方式来链接它们。

比如，现在我们有了一个 C 库文件，它的头文件是 **f.h**，产生的 lib（库）文件是 f.lib，那么我们如果要在 C++中使用这个库文件，我们需要这样写:

```
extern "C"
{
＃i nclude "f.h"
}
```

回到上面的问题，如果要改正链接错误，我们需要这样子改写 test.cxx:

```
extern "C"
{
extern void f1();
}

int main()
{
f1();

return 0;
}
```

重新编译并且链接就可以过去了.

总结

C 和 C++对函数的处理方式是不同的.extern "C"是使 C++能够调用 C 写作的库文件的一个手段，如果要对编译器提示使用 C 的方式来处理函数的话，那么就要使用 extern "C"来说明。

```c
/****************************************Copyright (c)****************************************************
** File name:          target.c     目标设定

#define IN_TARGET
#include "config.h"
/*********************************************************************************************************
** Function name:          IRQ_Exception
        void __irq IRQ_Exception(void)         // IRQ中断异常事件，本例死循环
{
    while(1);                    //这一句替换为自己的代码
}
/*********************************************************************************************************
** Function name:          FIQ_Exception
        void FIQ_Exception(void)               // FIQ中断异常事件，本例死循环
{
    while(1);                    //这一句替换为自己的代码
}
/*********************************************************************************************************
** Function name:          TargetInit
void TargetInit(void)
   {
    /* 添加自己的代码*/
   }
        void TargetResetInit(void)  // 目标复位初始化
  {
#ifdef __DEBUG                          // 采用调试方式进行编译,书本166页说明
    MEMMAP = 0x3;                       // MEMMAP的MAP[1:0] = 3 ,采用外包存储器进行重新映射
    #endif
#ifdef __OUT_CHIP                       // 采用外部芯片存储器
    MEMMAP = 0x3;                       // MEMMAP的MAP[1:0] = 3 ,采用外包存储器进行重新映射
#endif
#ifdef __IN_CHIP                        // 采用芯片内部存储器
    MEMMAP = 0x1;                       // MEMMAP的MAP[1:0] = 1 ,采用用户FLASH进行重新映射
#endif
/* 设置系统各部分时钟*/
// 设置系统各部分时钟，由系统硬件决定
// 设置外设时钟（VPB时钟pclk）与系统时钟（cclk）的分频比
// pclk与cclk具体值得大小在config.h中定义
    PLLCON = 1;                                 // 设置激活但未连接PLL
#if (Fpclk / (Fcclk / 4)) == 1                  // 此值由系统硬件决定
    VPBDIV = 0;                                 // VPB时钟频率Fpclk只能为系统时钟频率Fcclk的1、2、4倍
#endif
#if (Fpclk / (Fcclk / 4)) == 2
    VPBDIV = 2;
#endif
#if (Fpclk / (Fcclk / 4)) == 4
    VPBDIV = 1;
#endif
//设定PLL的乘因子M和除因子P的值，Fcclk / Fosc = M, Fcco / Fcclk = 2 * P
#if (Fcco / Fcclk) == 2                         // CCO频率，必须为Fcclk的2、4、8、16倍，范围156～320MHz
    PLLCFG = ((Fcclk / Fosc) - 1) | (0 << 5);
#endif
#if (Fcco / Fcclk) == 4
    PLLCFG = ((Fcclk / Fosc) - 1) | (1 << 5);
#endif
#if (Fcco / Fcclk) == 8
    PLLCFG = ((Fcclk / Fosc) - 1) | (2 << 5);
#endif
#if (Fcco / Fcclk) == 16
    PLLCFG = ((Fcclk / Fosc) - 1) | (3 << 5);
#endif
```

```
    PLLFEED = 0xaa;                     // 发送PLL馈送序列，执行设定PLL的动作
    PLLFEED = 0x55;
    while((PLLSTAT & (1 << 10)) == 0);  // 等待PLL锁定
    PLLCON = 3;                         // 设置激活并连接PLL
    PLLFEED = 0xaa;           // 发送PLL馈送序列，执行激活和连接动作
    PLLFEED = 0x55;
    /* 设置存储器加速模块*/
    MAMCR = 0;                // 关闭存储加速器MAM（00禁止/01部分使能/10完全使能）
#if Fcclk < 20000000          // 系统时钟是否低于20MHz，建议置1
    MAMTIM = 1;
#else
#if Fcclk < 40000000          // 系统时钟是否在20～40MHz之间，建议置2
    MAMTIM = 2;
#else
    MAMTIM = 3;               // 系统时钟是否高于40MHz，建议置3
#endif
#endif
    MAMCR = 2;                // 完全使能存储加速器MAM
    /* 初始化VIC */
    VICIntEnClr = 0xffffffff; // 中断使能清零寄存器
    VICVectAddr = 0;          // 向量地址寄存器
    VICIntSelect = 0;         // 设置所有中断分配为IRQ中断（中断选择寄存器）
    /* 添加自己的代码*/
   }
/*********************************************************************************************************
**                  以下为一些与系统相关的库函数的实现
**                  具体作用请ads的参考编译器与库函数手册
**                  用户可以根据自己的要求修改
********************************************************************************************************/
#include "rt_sys.h"
#include "stdio.h"

// (__use_no_semihosting_swi)这段语句是编译器预定义的，就是不使用编译器自带的软中断服务函数。

//该软中断的中断号为0x123456，用来实现一些调试信息和系统调用。具体参看相关的编译器手册

#pragma import(__use_no_semihosting_swi)
#pragma import(__use_two_region_memory)

int __rt_div0(int a)
    { a = a;   return 0;}
int fputc(int ch,FILE *f)
    { ch = ch; f = f;    return 0;}
int fgetc(FILE *f)
    { f = f;   return 0;}
int _sys_close(FILEHANDLE fh)
    { fh = fh;    return 0;}
int _sys_write(FILEHANDLE fh, const unsigned char * buf,unsigned len, int mode)
   {
      fh = fh;
      buf = buf;
      len =len;
      mode = mode;
      return 0;
   }
int _sys_read(FILEHANDLE fh, unsigned char * buf,unsigned len, int mode)
   {
      fh = fh;
      buf = buf;
      len =len;
      mode = mode;
      return 0;
```

```
    }
void _ttywrch(int ch)
   { ch = ch;}
int _sys_istty(FILEHANDLE fh)
   { fh = fh;    return 0;}
int _sys_seek(FILEHANDLE fh, long pos)
   { fh = fh;    return 0;}
int _sys_ensure(FILEHANDLE fh)
   { fh = fh;    return 0;}
long _sys_flen(FILEHANDLE fh)
   { fh = fh;    return 0;}
int _sys_tmpnam(char * name, int sig, unsigned maxlen)
   {
    name = name;
    sig = sig;
    maxlen = maxlen;
    return 0;
   }
void _sys_exit(int returncode)
   { returncode = returncode;}
char *_sys_command_string(char * cmd, int len)
   { cmd = cmd;    len = len;    return 0;}
/*****************************************************************************************************
```

```
/*****************************************Copyright (c)**************************************************
** File Name: config.h  配置文件，设定数据类型，ARM芯片的配置及时钟等
#ifndef __CONFIG_H
#define __CONFIG_H

#ifndef TRUE                                    // 设定TRUE为1
#define TRUE  1
#endif

#ifndef FALSE                                   // 设定FALSE为0
#define FALSE 0
#endif

typedef unsigned char  uint8;                   // 无符号位整型变量
typedef signed   char  int8;                    // 有符号位整型变量
typedef unsigned short uint16;                  // 无符号位整型变量
typedef signed   short int16;                   // 有符号位整型变量
typedef unsigned int   uint32;                  // 无符号位整型变量
typedef signed   int   int32;                   // 有符号位整型变量
typedef float          fp32;                    // 单精度浮点数（位长度）
typedef double         fp64;                    // 双精度浮点数（位长度）

/*********************************  ARM的特殊代码 /**********************************************************/
//这一段无需改动
#include    "LPC2294.h"                        //包含ARM芯片寄存器定义及固件程序定义
//应用程序配置
//以下根据需要改动

//本例子的配置
// 系统设置, Fosc、Fcclk、Fcco、Fpclk必须定义
#define Fosc            11059200              // 应当与实际一至晶振频率,10MHz~25MHz，应当与实际一至
#define Fcclk           (Fosc * 4)            // 系统频率，必须为Fosc的整数倍(1~32)，且<=60MHZ
#define Fcco            (Fcclk * 4)           // CCO频率，必须为Fcclk的、、、倍，范围为MHz~320MHz
#define Fpclk           (Fcclk / 4) * 1       // VPB时钟频率，只能为(Fcclk / 4)的、、倍
#include    "target.h"                        // 包含目标文件（这一句不能删除）
#endif
/**********************************************************************************************************/
```

```
/****************************************Copyright (c)**************************************************
** File name:          LPC2294.h
** Descriptions:       详细lpc22xx\lpc212x\lpc211x\lpc210x描述专用寄存器及固件函数
/* 外部总线控制器  EXTERNAL MEMORY CONTROLLER (EMC) */ /* lpc22xx only */存储器组0/1/2/3 的配置寄存器RW
#define BCFG0          (*((volatile unsigned int *) 0xFFE00000))       //8000 0000－80FF FFFF
#define BCFG1          (*((volatile unsigned int *) 0xFFE00004))       //8100 0000－81FF FFFF
#define BCFG2          (*((volatile unsigned int *) 0xFFE00008))       //8200 0000－82FF FFFF
#define BCFG3          (*((volatile unsigned int *) 0xFFE0000C))       //8300 0000－83FF FFFF
/* 外部中断控制寄存器  External Interrupts */               /* Not used for lpc210x*/
#define EXTINT         (*((volatile unsigned char *) 0xE01FC140))     //外部中断标志寄存器RW
#define EXTWAKE        (*((volatile unsigned char *) 0xE01FC144))     //外部中断唤醒寄存器RW
#define EXTMODE        (*((volatile unsigned char *) 0xE01FC148))     //外部中断方式寄存器RW
#define EXTPOLAR       (*((volatile unsigned char *) 0xE01FC14C))     //外部中断极性寄存器RW
/* 内存remap控制寄存器 SMemory mapping control */
#define MEMMAP         (*((volatile unsigned char *) 0xE01FC040))     //存储器映射控制寄存器(boot、ram、flash、sram)
/* PLL控制寄存器 Phase Locked Loop (PLL)*/
#define PLLCON         (*((volatile unsigned char *) 0xE01FC080))     //PLL 控制寄存器RW
#define PLLCFG         (*((volatile unsigned char *) 0xE01FC084))     //PLL 配置寄存器RW
#define PLLSTAT        (*((volatile unsigned short*) 0xE01FC088))     //PLL 状态寄存器RO
#define PLLFEED        (*((volatile unsigned char *) 0xE01FC08C))     //PLL 馈送寄存器WO
/* 功率控制寄存器 Power Control */
#define PCON           (*((volatile unsigned char *) 0xE01FC0C0))     //功率控制寄存器（节电模式）RW
#define PCONP          (*((volatile unsigned long *) 0xE01FC0C4))     //外设功率控制寄存器（关闭外设节、电）RW
/* VLSI外设总线（VPB）分频寄存器 VPB Divider */
#define VPBDIV         (*((volatile unsigned char *) 0xE01FC100))     //控制VPB 时钟速率与处理器时钟之间的关系RW
/* 存储器加速模块 Memory Accelerator Module (MAM)*/
#define MAMCR          (*((volatile unsigned char *) 0xE01FC000))     //存储器加速器模块控制寄存器RW
#define MAMTIM         (*((volatile unsigned char *) 0xE01FC004))     //存储器加速器定时控制RW
/* 向量中断控制器(VIC)的特殊寄存器 Vectored Interrupt Controller (VIC)*/
#define VICIRQStatus    (*((volatile unsigned long *) 0xFFFFF000))     //IRQ 状态寄存RO
#define VICFIQStatus    (*((volatile unsigned long *) 0xFFFFF004))     //FIQ 状态寄存器RO
#define VICRawIntr      (*((volatile unsigned long *) 0xFFFFF008))     //所有中断的状态寄存器RO
#define VICIntSelect    (*((volatile unsigned long *) 0xFFFFF00C))     //中断选择寄存器(分配TRQ/FIQ)RW
#define VICIntEnable    (*((volatile unsigned long *) 0xFFFFF010))     //中断使能寄存器RW
#define VICIntEnClr     (*((volatile unsigned long *) 0xFFFFF014))     //中断使能清零寄存器RO
#define VICSoftInt      (*((volatile unsigned long *) 0xFFFFF018))     //软件中断寄存器RW
#define VICSoftIntClear (*((volatile unsigned long *) 0xFFFFF01C))     //软件中断清零寄存器WO
#define VICProtection   (*((volatile unsigned long *) 0xFFFFF020))     //保护使能寄存器RW
#define VICVectAddr     (*((volatile unsigned long *) 0xFFFFF030))     //向量地址寄存器(IRQ中断)RW
#define VICDefVectAddr  (*((volatile unsigned long *) 0xFFFFF034))     //默认向量地址寄存器(非IRQ中断)RW
#define VICVectAddr0    (*((volatile unsigned long *) 0xFFFFF100))     //向量地址0 寄存器RW
#define VICVectAddr1    (*((volatile unsigned long *) 0xFFFFF104))     //向量地址1 寄存器RW
#define VICVectAddr2    (*((volatile unsigned long *) 0xFFFFF108))     //向量地址2 寄存器RW
#define VICVectAddr3    (*((volatile unsigned long *) 0xFFFFF10C))     //向量地址3 寄存器RW
#define VICVectAddr4    (*((volatile unsigned long *) 0xFFFFF110))     //向量地址4 寄存器RW
#define VICVectAddr5    (*((volatile unsigned long *) 0xFFFFF114))     //向量地址5 寄存器RW
#define VICVectAddr6    (*((volatile unsigned long *) 0xFFFFF118))     //向量地址6 寄存器RW
#define VICVectAddr7    (*((volatile unsigned long *) 0xFFFFF11C))     //向量地址7 寄存器RW
#define VICVectAddr8    (*((volatile unsigned long *) 0xFFFFF120))     //向量地址8 寄存器RW
#define VICVectAddr9    (*((volatile unsigned long *) 0xFFFFF124))     //向量地址9 寄存器RW
#define VICVectAddr10   (*((volatile unsigned long *) 0xFFFFF128))     //向量地址10 寄存器RW
#define VICVectAddr11   (*((volatile unsigned long *) 0xFFFFF12C))     //向量地址11 寄存器RW
#define VICVectAddr12   (*((volatile unsigned long *) 0xFFFFF130))     //向量地址12 寄存器RW
#define VICVectAddr13   (*((volatile unsigned long *) 0xFFFFF134))     //向量地址13 寄存器RW
#define VICVectAddr14   (*((volatile unsigned long *) 0xFFFFF138))     //向量地址14 寄存器RW
#define VICVectAddr15   (*((volatile unsigned long *) 0xFFFFF13C))     //向量地址15 寄存器RW
#define VICVectCntl0    (*((volatile unsigned long *) 0xFFFFF200))     //向量控制0 寄存器RW
#define VICVectCntl1    (*((volatile unsigned long *) 0xFFFFF204))     //向量控制1 寄存器RW
```

```
#define VICVectCntl2     (*((volatile unsigned long *) 0xFFFFF208))     //向量控制2 寄存器RW
#define VICVectCntl3     (*((volatile unsigned long *) 0xFFFFF20C))     //向量控制3 寄存器RW
#define VICVectCntl4     (*((volatile unsigned long *) 0xFFFFF210))     //向量控制4 寄存器RW
#define VICVectCntl5     (*((volatile unsigned long *) 0xFFFFF214))     //向量控制5 寄存器RW
#define VICVectCntl6     (*((volatile unsigned long *) 0xFFFFF218))     //向量控制6 寄存器RW
#define VICVectCntl7     (*((volatile unsigned long *) 0xFFFFF21C))     //向量控制7 寄存器RW
#define VICVectCntl8     (*((volatile unsigned long *) 0xFFFFF220))     //向量控制8 寄存器RW
#define VICVectCntl9     (*((volatile unsigned long *) 0xFFFFF224))     //向量控制9 寄存器RW
#define VICVectCntl10    (*((volatile unsigned long *) 0xFFFFF228))     //向量控制10 寄存器RW
#define VICVectCntl11    (*((volatile unsigned long *) 0xFFFFF22C))     //向量控制11 寄存器RW
#define VICVectCntl12    (*((volatile unsigned long *) 0xFFFFF230))     //向量控制12 寄存器RW
#define VICVectCntl13    (*((volatile unsigned long *) 0xFFFFF234))     //向量控制13 寄存器RW
#define VICVectCntl14    (*((volatile unsigned long *) 0xFFFFF238))     //向量控制14 寄存器RW
#define VICVectCntl15    (*((volatile unsigned long *) 0xFFFFF23C))     //向量控制15 寄存器RW
/* 管脚连接模块控制寄存器 Pin Connect Block */
#define PINSEL0          (*((volatile unsigned long *) 0xE002C000))     //管脚功能选择寄存器0RW
#define PINSEL1          (*((volatile unsigned long *) 0xE002C004))     //管脚功能选择寄存器1RW
                                                                    /* Not used for lpc210x*/
#define PINSEL2          (*((volatile unsigned long *) 0xE002C014))     //管脚功能选择寄存器2RW
/* 通用并行IO口的特殊寄存器 General Purpose Input/Output (GPIO)*/  /* lpc210x only */
#define IOPIN            (*((volatile unsigned long *) 0xE0028000))     //GPIO 管脚值寄存器RO
#define IOSET            (*((volatile unsigned long *) 0xE0028004))     //GPIO 输出置位寄存（读/置位）
#define IODIR            (*((volatile unsigned long *) 0xE0028008))     //GPIO 方向控制寄存RW
#define IOCLR            (*((volatile unsigned long *) 0xE002800C))     //GPIO 输出清零寄存（只清零）
                                                                   /* Not used for lpc210x*/
#define IO0PIN           (*((volatile unsigned long *) 0xE0028000))     //GPIO0 管脚值寄存器RO
#define IO0SET           (*((volatile unsigned long *) 0xE0028004))     //GPIO0 输出置位寄存（读/置位）
#define IO0DIR           (*((volatile unsigned long *) 0xE0028008))     //GPIO0 方向控制寄存RW
#define IO0CLR           (*((volatile unsigned long *) 0xE002800C))     //GPIO0 输出清零寄存（只清零

#define IO1PIN           (*((volatile unsigned long *) 0xE0028010))     //GPIO1 管脚值寄存器RO
#define IO1SET           (*((volatile unsigned long *) 0xE0028014))     //GPIO1 输出置位寄存（读/置位）
#define IO1DIR           (*((volatile unsigned long *) 0xE0028018))     //GPIO1 方向控制寄存RW
#define IO1CLR           (*((volatile unsigned long *) 0xE002801C))     //GPIO1 输出清零寄存（只清零
                                                                   /* lpc22xx only */
#define IO2PIN           (*((volatile unsigned long *) 0xE0028020))     //GPIO2 管脚值寄存器RO
#define IO2SET           (*((volatile unsigned long *) 0xE0028024))     //GPIO2 输出置位寄存（读/置位）
#define IO2DIR           (*((volatile unsigned long *) 0xE0028028))     //GPIO2 方向控制寄存RW
#define IO2CLR           (*((volatile unsigned long *) 0xE002802C))     //GPIO2 输出清零寄存（只清零

#define IO3PIN           (*((volatile unsigned long *) 0xE0028030))     //GPIO3 管脚值寄存器RO
#define IO3SET           (*((volatile unsigned long *) 0xE0028034))     //GPIO3 输出置位寄存（读/置位）
#define IO3DIR           (*((volatile unsigned long *) 0xE0028038))     //GPIO3 方向控制寄存RW
#define IO3CLR           (*((volatile unsigned long *) 0xE002803C))     //GPIO3 输出清零寄存（只清零
/* 通用异步串行口(UART0)的特殊寄存器 Universal Asynchronous Receiver Transmitter 0 (UART0)/
#define U0RBR            (*((volatile unsigned char *) 0xE000C000))     //UART0 接收器缓存寄存器RO
#define U0THR            (*((volatile unsigned char *) 0xE000C000))     //UART0 发送器保持寄存器WO
#define U0IER            (*((volatile unsigned char *) 0xE000C004))     //UART0 中断使能寄存器RW
#define U0IIR            (*((volatile unsigned char *) 0xE000C008))     //UART0 中断标识寄存器RO
#define U0FCR            (*((volatile unsigned char *) 0xE000C008))     //UART0 FIFO 控制寄存器WO
#define U0LCR            (*((volatile unsigned char *) 0xE000C00C))     //UART0 线控制寄存器RW
#define U0LSR            (*((volatile unsigned char *) 0xE000C014))     //UART0 线状态寄存器RO
#define U0SCR            (*((volatile unsigned char *) 0xE000C01C))     //UART0 高速缓存寄存器RW
#define U0DLL            (*((volatile unsigned char *) 0xE000C000))     //UART0 除数锁存LSB 寄存器RW
#define U0DLM            (*((volatile unsigned char *) 0xE000C004))     //UART0 除数锁存MSB 寄存器RW
 /* 通用异步串行口(UART1)的特殊寄存器 Universal Asynchronous Receiver Transmitter 1 (UART1)*/
#define U1RBR            (*((volatile unsigned char *) 0xE0010000))     //UART1 接收器缓存寄存器RO
#define U1THR            (*((volatile unsigned char *) 0xE0010000))     //UART1 发送器保持寄存器WO
#define U1IER            (*((volatile unsigned char *) 0xE0010004))     //UART1 中断使能寄存器RW
#define U1IIR            (*((volatile unsigned char *) 0xE0010008))     //UART1 中断标识寄存器RO
```

```
#define U1FCR          (*((volatile unsigned char *) 0xE0010008))     //UART1 FIFO 控制寄存器WO
#define U1LCR          (*((volatile unsigned char *) 0xE001000C))     //UART1 线控制寄存器RW
#define U1MCR          (*((volatile unsigned char *) 0xE0010010))     //Modem控制(与Uart0区别处)RW
#define U1LSR          (*((volatile unsigned char *) 0xE0010014))     //UART1 线状态寄存器RO
#define U1MSR          (*((volatile unsigned char *) 0xE0010018))     //Modem 状态RO
#define U1SCR          (*((volatile unsigned char *) 0xE001001C))     //UART1 高速缓存寄存器RW
#define U1DLL          (*((volatile unsigned char *) 0xE0010000))     //UART1 除数锁存LSB 寄存器RW
#define U1DLM          (*((volatile unsigned char *) 0xE0010004))     //UART1 除数锁存MSB 寄存器RW
/* 芯片间总线（I2C）的特殊寄存器 I2C (8/16 bit data bus)*/
#define I2CONSET       (*((volatile unsigned char *) 0xE001C000))     //I2C 控制置位寄存器（读/置位）
#define I2STAT         (*((volatile unsigned char *) 0xE001C004))     //I2C 状态寄存器RO
#define I2DAT          (*((volatile unsigned char *) 0xE001C008))     //I2C 数据寄存器RW
#define I2ADR          (*((volatile unsigned char *) 0xE001C00C))     //I2C 从地址寄存器RW
#define I2SCLH         (*((volatile unsigned short *) 0xE001C010))    //I2C SCL 占空比寄存器高半字RW
#define I2SCLL         (*((volatile unsigned short *) 0xE001C014))    //I2C SCL 占空比寄存器低半字RW
#define I2CONCLR       (*((volatile unsigned char *) 0xE001C018))     //I2C 控制清零寄存器（只清零）
/* SPI总线接口的特殊寄存器SPI (Serial Peripheral Interface)*/ /* only for lpc210x*/
#define SPI_SPCR       (*((volatile unsigned char *) 0xE0020000))     //SPI 控制寄存器
#define SPI_SPSR       (*((volatile unsigned char *) 0xE0020004))     //SPI 状态寄存器
#define SPI_SPDR       (*((volatile unsigned char *) 0xE0020008))     //SPI 数据寄存器
#define SPI_SPCCR      (*((volatile unsigned char *) 0xE002000C))     //SPI 时钟计数寄存器
#define SPI_SPINT      (*((volatile unsigned char *) 0xE002001C))     //SPI 中断标志寄存器
                                                              /* Not used for lpc210x*/
#define S0PCR          (*((volatile unsigned char *) 0xE0020000))     //SPI0 控制寄存器
#define S0PSR          (*((volatile unsigned char *) 0xE0020004))     //SPI0 状态寄存器
#define S0PDR          (*((volatile unsigned char *) 0xE0020008))     //SPI0 数据寄存器
#define S0PCCR         (*((volatile unsigned char *) 0xE002000C))     //SPI0 时钟计数寄存器
#define S0PINT         (*((volatile unsigned char *) 0xE002001C))     //SPI0 中断标志寄存器

#define S1PCR          (*((volatile unsigned char *) 0xE0030000))     //SPI1 控制寄存器
#define S1PSR          (*((volatile unsigned char *) 0xE0030004))     //SPI1 状态寄存器
#define S1PDR          (*((volatile unsigned char *) 0xE0030008))     //SPI1 数据寄存器
#define S1PCCR         (*((volatile unsigned char *) 0xE003000C))     //SPI1 时钟计数寄存器
#define S1PINT         (*((volatile unsigned char *) 0xE003001C))     //SPI1 中断标志寄存器
/* CAN控制器和接收路波器 CAN CONTROLLERS AND ACCEPTANCE FILTER */ /* lpc2119\2129\2292\2294 only */
#define CAN1MOD        (*((volatile unsigned long *) 0xE0044000))     //CAN 控制器的工作模式
#define CAN1CMR        (*((volatile unsigned long *) 0xE0044004))     //CAN 命令寄存器
#define CAN1GSR        (*((volatile unsigned long *) 0xE0044008))     //全局状态控制器
#define CAN1ICR        (*((volatile unsigned long *) 0xE004400C))     //中断状态，仲裁丢失捕获，错误码捕获寄存器
#define CAN1IER        (*((volatile unsigned long *) 0xE0044010))     //中断使能寄存器
#define CAN1BTR        (*((volatile unsigned long *) 0xE0044014))     //总线时序寄存器
#define CAN1EWL        (*((volatile unsigned long *) 0xE0044018))     //出错警告界限寄存器
#define CAN1SR         (*((volatile unsigned long *) 0xE004401C))     //状态寄存器
#define CAN1RFS        (*((volatile unsigned long *) 0xE0044020))     //接收帧状态寄存器
#define CAN1RID        (*((volatile unsigned long *) 0xE0044024))     //接收标识符寄存器
#define CAN1RDA        (*((volatile unsigned long *) 0xE0044028))     //接收到的数据字节1-4
#define CAN1RDB        (*((volatile unsigned long *) 0xE004402C))     //接收到的数据字节5-8
#define CAN1TFI1       (*((volatile unsigned long *) 0xE0044030))     //发送帧信息(1)
#define CAN1TID1       (*((volatile unsigned long *) 0xE0044034))     //发送标识符(1)
#define CAN1TDA1       (*((volatile unsigned long *) 0xE0044038))     //发送数据字节1-4(1)
#define CAN1TDB1       (*((volatile unsigned long *) 0xE004403C))     //发送数据字节5-8(1)
#define CAN1TFI2       (*((volatile unsigned long *) 0xE0044040))     //发送帧信息(2)
#define CAN1TID2       (*((volatile unsigned long *) 0xE0044044))     //发送标识符(2)
#define CAN1TDA2       (*((volatile unsigned long *) 0xE0044048))     //发送数据字节1-4(2)
#define CAN1TDB2       (*((volatile unsigned long *) 0xE004404C))     //发送数据字节5-8(2)
#define CAN1TFI3       (*((volatile unsigned long *) 0xE0044050))     //发送帧信息(3)
#define CAN1TID3       (*((volatile unsigned long *) 0xE0044054))     //发送标识符(3)
#define CAN1TDA3       (*((volatile unsigned long *) 0xE0044058))     //发送数据字节1-4(3)
#define CAN1TDB3       (*((volatile unsigned long *) 0xE004405C))     //发送数据字节5-8(3)
   //参考CAN1
#define CAN2MOD        (*((volatile unsigned long *) 0xE0048000))      /* lpc2119\2129\2292\2294 only */
```

```
#define CAN2CMR          (*((volatile unsigned long *) 0xE0048004))      /* lpc2119\2129\2292\2294 only */
#define CAN2GSR          (*((volatile unsigned long *) 0xE0048008))      /* lpc2119\2129\2292\2294 only */
#define CAN2ICR          (*((volatile unsigned long *) 0xE004800C))      /* lpc2119\2129\2292\2294 only */
#define CAN2IER          (*((volatile unsigned long *) 0xE0048010))      /* lpc2119\2129\2292\2294 only */
#define CAN2BTR          (*((volatile unsigned long *) 0xE0048014))      /* lpc2119\2129\2292\2294 only */
#define CAN2EWL          (*((volatile unsigned long *) 0xE004801C))      /* lpc2119\2129\2292\2294 only */
#define CAN2SR           (*((volatile unsigned long *) 0xE0048020))      /* lpc2119\2129\2292\2294 only */
#define CAN2RFS          (*((volatile unsigned long *) 0xE0048024))      /* lpc2119\2129\2292\2294 only */
#define CAN2RDA          (*((volatile unsigned long *) 0xE0048028))      /* lpc2119\2129\2292\2294 only */
#define CAN2RDB          (*((volatile unsigned long *) 0xE004802C))      /* lpc2119\2129\2292\2294 only */
#define CAN2TFI1         (*((volatile unsigned long *) 0xE0048030))      /* lpc2119\2129\2292\2294 only */
#define CAN2TID1         (*((volatile unsigned long *) 0xE0048034))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDA1         (*((volatile unsigned long *) 0xE0048038))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDB1         (*((volatile unsigned long *) 0xE004803C))      /* lpc2119\2129\2292\2294 only */
#define CAN2TFI2         (*((volatile unsigned long *) 0xE0048040))      /* lpc2119\2129\2292\2294 only */
#define CAN2TID2         (*((volatile unsigned long *) 0xE0048044))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDA2         (*((volatile unsigned long *) 0xE0048048))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDB2         (*((volatile unsigned long *) 0xE004804C))      /* lpc2119\2129\2292\2294 only */
#define CAN2TFI3         (*((volatile unsigned long *) 0xE0048050))      /* lpc2119\2129\2292\2294 only */
#define CAN2TID3         (*((volatile unsigned long *) 0xE0048054))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDA3         (*((volatile unsigned long *) 0xE0048058))      /* lpc2119\2129\2292\2294 only */
#define CAN2TDB3         (*((volatile unsigned long *) 0xE004805C))      /* lpc2119\2129\2292\2294 only */
    //参考CAN1
#define CAN3MOD          (*((volatile unsigned long *) 0xE004C000))      /* lpc2119\2129\2292\2294 only */
#define CAN3CMR          (*((volatile unsigned long *) 0xE004C004))      /* lpc2119\2129\2292\2294 only */
#define CAN3GSR          (*((volatile unsigned long *) 0xE004C008))      /* lpc2119\2129\2292\2294 only */
#define CAN3ICR          (*((volatile unsigned long *) 0xE004C00C))      /* lpc2119\2129\2292\2294 only */
#define CAN3IER          (*((volatile unsigned long *) 0xE004C010))      /* lpc2119\2129\2292\2294 only */
#define CAN3BTR          (*((volatile unsigned long *) 0xE004C014))      /* lpc2119\2129\2292\2294 only */
#define CAN3EWL          (*((volatile unsigned long *) 0xE004C01C))      /* lpc2119\2129\2292\2294 only */
#define CAN3SR           (*((volatile unsigned long *) 0xE004C020))      /* lpc2119\2129\2292\2294 only */
#define CAN3RFS          (*((volatile unsigned long *) 0xE004C024))      /* lpc2119\2129\2292\2294 only */
#define CAN3RDA          (*((volatile unsigned long *) 0xE004C028))      /* lpc2119\2129\2292\2294 only */
#define CAN3RDB          (*((volatile unsigned long *) 0xE004C02C))      /* lpc2119\2129\2292\2294 only */
#define CAN3TFI1         (*((volatile unsigned long *) 0xE004C030))      /* lpc2119\2129\2292\2294 only */
#define CAN3TID1         (*((volatile unsigned long *) 0xE004C034))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDA1         (*((volatile unsigned long *) 0xE004C038))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDB1         (*((volatile unsigned long *) 0xE004C03C))      /* lpc2119\2129\2292\2294 only */
#define CAN3TFI2         (*((volatile unsigned long *) 0xE004C040))      /* lpc2119\2129\2292\2294 only */
#define CAN3TID2         (*((volatile unsigned long *) 0xE004C044))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDA2         (*((volatile unsigned long *) 0xE004C048))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDB2         (*((volatile unsigned long *) 0xE004C04C))      /* lpc2119\2129\2292\2294 only */
#define CAN3TFI3         (*((volatile unsigned long *) 0xE004C050))      /* lpc2119\2129\2292\2294 only */
#define CAN3TID3         (*((volatile unsigned long *) 0xE004C054))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDA3         (*((volatile unsigned long *) 0xE004C058))      /* lpc2119\2129\2292\2294 only */
#define CAN3TDB3         (*((volatile unsigned long *) 0xE004C05C))      /* lpc2119\2129\2292\2294 only */
    //参考CAN1
#define CAN4MOD          (*((volatile unsigned long *) 0xE0050000))      /* lpc2119\2129\2292\2294 only */
#define CAN4CMR          (*((volatile unsigned long *) 0xE0050004))      /* lpc2119\2129\2292\2294 only */
#define CAN4GSR          (*((volatile unsigned long *) 0xE0050008))      /* lpc2119\2129\2292\2294 only */
#define CAN4ICR          (*((volatile unsigned long *) 0xE005000C))      /* lpc2119\2129\2292\2294 only */
#define CAN4IER          (*((volatile unsigned long *) 0xE0050010))      /* lpc2119\2129\2292\2294 only */
#define CAN4BTR          (*((volatile unsigned long *) 0xE0050014))      /* lpc2119\2129\2292\2294 only */
#define CAN4EWL          (*((volatile unsigned long *) 0xE005001C))      /* lpc2119\2129\2292\2294 only */
#define CAN4SR           (*((volatile unsigned long *) 0xE0050020))      /* lpc2119\2129\2292\2294 only */
#define CAN4RFS          (*((volatile unsigned long *) 0xE0050024))      /* lpc2119\2129\2292\2294 only */
#define CAN4RDA          (*((volatile unsigned long *) 0xE0050028))      /* lpc2119\2129\2292\2294 only */
#define CAN4RDB          (*((volatile unsigned long *) 0xE005002C))      /* lpc2119\2129\2292\2294 only */
#define CAN4TFI1         (*((volatile unsigned long *) 0xE0050030))      /* lpc2119\2129\2292\2294 only */
#define CAN4TID1         (*((volatile unsigned long *) 0xE0050034))      /* lpc2119\2129\2292\2294 only */
#define CAN4TDA1         (*((volatile unsigned long *) 0xE0050038))      /* lpc2119\2129\2292\2294 only */
```

```
#define CAN4TDB1        (*((volatile unsigned long *) 0xE005003C))      /* lpc2119\2129\2292\2294 only */
#define CAN4TFI2        (*((volatile unsigned long *) 0xE0050040))      /* lpc2119\2129\2292\2294 only */
#define CAN4TID2        (*((volatile unsigned long *) 0xE0050044))      /* lpc2119\2129\2292\2294 only */
#define CAN4TDA2        (*((volatile unsigned long *) 0xE0050048))      /* lpc2119\2129\2292\2294 only */
#define CAN4TDB2        (*((volatile unsigned long *) 0xE005004C))      /* lpc2119\2129\2292\2294 only */
#define CAN4TFI3        (*((volatile unsigned long *) 0xE0050050))      /* lpc2119\2129\2292\2294 only */
#define CAN4TID3        (*((volatile unsigned long *) 0xE0050054))      /* lpc2119\2129\2292\2294 only */
#define CAN4TDA3        (*((volatile unsigned long *) 0xE0050058))      /* lpc2119\2129\2292\2294 only */
#define CAN4TDB3        (*((volatile unsigned long *) 0xE005005C))      /* lpc2119\2129\2292\2294 only */
    //参考CAN1
#define CAN5MOD         (*((volatile unsigned long *) 0xE0054000))      /* lpc2119\2129\2292\2294 only */
#define CAN5CMR         (*((volatile unsigned long *) 0xE0054004))      /* lpc2119\2129\2292\2294 only */
#define CAN5GSR         (*((volatile unsigned long *) 0xE0054008))      /* lpc2119\2129\2292\2294 only */
#define CAN5ICR         (*((volatile unsigned long *) 0xE005400C))      /* lpc2119\2129\2292\2294 only */
#define CAN5IER         (*((volatile unsigned long *) 0xE0054010))      /* lpc2119\2129\2292\2294 only */
#define CAN5BTR         (*((volatile unsigned long *) 0xE0054014))      /* lpc2119\2129\2292\2294 only */
#define CAN5EWL         (*((volatile unsigned long *) 0xE005401C))      /* lpc2119\2129\2292\2294 only */
#define CAN5SR          (*((volatile unsigned long *) 0xE0054020))      /* lpc2119\2129\2292\2294 only */
#define CAN5RFS         (*((volatile unsigned long *) 0xE0054024))      /* lpc2119\2129\2292\2294 only */
#define CAN5RDA         (*((volatile unsigned long *) 0xE0054028))      /* lpc2119\2129\2292\2294 only */
#define CAN5RDB         (*((volatile unsigned long *) 0xE005402C))      /* lpc2119\2129\2292\2294 only */
#define CAN5TFI1        (*((volatile unsigned long *) 0xE0054030))      /* lpc2119\2129\2292\2294 only */
#define CAN5TID1        (*((volatile unsigned long *) 0xE0054034))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDA1        (*((volatile unsigned long *) 0xE0054038))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDB1        (*((volatile unsigned long *) 0xE005403C))      /* lpc2119\2129\2292\2294 only */
#define CAN5TFI2        (*((volatile unsigned long *) 0xE0054040))      /* lpc2119\2129\2292\2294 only */
#define CAN5TID2        (*((volatile unsigned long *) 0xE0054044))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDA2        (*((volatile unsigned long *) 0xE0054048))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDB2        (*((volatile unsigned long *) 0xE005404C))      /* lpc2119\2129\2292\2294 only */
#define CAN5TFI3        (*((volatile unsigned long *) 0xE0054050))      /* lpc2119\2129\2292\2294 only */
#define CAN5TID3        (*((volatile unsigned long *) 0xE0054054))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDA3        (*((volatile unsigned long *) 0xE0054058))      /* lpc2119\2129\2292\2294 only */
#define CAN5TDB3        (*((volatile unsigned long *) 0xE005405C))      /* lpc2119\2129\2292\2294 only */
    //参考CAN1
#define CAN6MOD         (*((volatile unsigned long *) 0xE0058000))      /* lpc2292\2294 only */
#define CAN6CMR         (*((volatile unsigned long *) 0xE0058004))      /* lpc2292\2294 only */
#define CAN6GSR         (*((volatile unsigned long *) 0xE0058008))      /* lpc2292\2294 only */
#define CAN6ICR         (*((volatile unsigned long *) 0xE005800C))      /* lpc2292\2294 only */
#define CAN6IER         (*((volatile unsigned long *) 0xE0058010))      /* lpc2292\2294 only */
#define CAN6BTR         (*((volatile unsigned long *) 0xE0058014))      /* lpc2292\2294 only */
#define CAN6EWL         (*((volatile unsigned long *) 0xE005801C))      /* lpc2292\2294 only */
#define CAN6SR          (*((volatile unsigned long *) 0xE0058020))      /* lpc2292\2294 only */
#define CAN6RFS         (*((volatile unsigned long *) 0xE0058024))      /* lpc2292\2294 only */
#define CAN6RDA         (*((volatile unsigned long *) 0xE0058028))      /* lpc2292\2294 only */
#define CAN6RDB         (*((volatile unsigned long *) 0xE005802C))      /* lpc2292\2294 only */
#define CAN6TFI1        (*((volatile unsigned long *) 0xE0058030))      /* lpc2292\2294 only */
#define CAN6TID1        (*((volatile unsigned long *) 0xE0058034))      /* lpc2292\2294 only */
#define CAN6TDA1        (*((volatile unsigned long *) 0xE0058038))      /* lpc2292\2294 only */
#define CAN6TDB1        (*((volatile unsigned long *) 0xE005803C))      /* lpc2292\2294 only */
#define CAN6TFI2        (*((volatile unsigned long *) 0xE0058040))      /* lpc2292\2294 only */
#define CAN6TID2        (*((volatile unsigned long *) 0xE0058044))      /* lpc2292\2294 only */
#define CAN6TDA2        (*((volatile unsigned long *) 0xE0058048))      /* lpc2292\2294 only */
#define CAN6TDB2        (*((volatile unsigned long *) 0xE005804C))      /* lpc2292\2294 only */
#define CAN6TFI3        (*((volatile unsigned long *) 0xE0058050))      /* lpc2292\2294 only */
#define CAN6TID3        (*((volatile unsigned long *) 0xE0058054))      /* lpc2292\2294 only */
#define CAN6TDA3        (*((volatile unsigned long *) 0xE0058058))      /* lpc2292\2294 only */
#define CAN6TDB3        (*((volatile unsigned long *) 0xE005805C))      /* lpc2292\2294 only */
                                            /* lpc2119\2129\2292\2294 only */
#define CANTxSR         (*((volatile unsigned long *) 0xE0040000))     //集中发送状态寄存器
#define CANRxSR         (*((volatile unsigned long *) 0xE0040004))     //集中接收状态寄存器
#define CANMSR          (*((volatile unsigned long *) 0xE0040008))     //集中其它状态寄存器
```

```
#define CANAFMR        (*((volatile unsigned long *) 0xE003C000))     //验收滤波器模式寄存器
#define CANSFF_sa      (*((volatile unsigned long *) 0xE003C004))     //标准帧单个起始地址寄存器
#define CANSFF_GRP_sa  (*((volatile unsigned long *) 0xE003C008))     //标准帧组起始地址寄存器
#define CANEFF_sa      (*((volatile unsigned long *) 0xE003C00C))     //扩展帧起始地址寄存器
#define CANEFF_GRP_sa  (*((volatile unsigned long *) 0xE003C010))     //扩展帧组起始地址寄存器
#define CANENDofTable  (*((volatile unsigned long *) 0xE003C014))     //AF 表格终止寄存器
#define CANLUTerrAd    (*((volatile unsigned long *) 0xE003C018))     //LUT 错误地址寄存器
#define CANLUTerr      (*((volatile unsigned long *) 0xE003C01C))     //LUT 错误寄存器
/* CAN Acceptance Filter RAM */
#define CANAFRAM       (*((volatile unsigned long *) 0xE0038000))     // ?
/* 定时器的特殊寄存器 Timer 0*/
#define T0IR           (*((volatile unsigned long *) 0xE0004000))     //中断寄存器
#define T0TCR          (*((volatile unsigned long *) 0xE0004004))     //定时器控制寄存器
#define T0TC           (*((volatile unsigned long *) 0xE0004008))     //定时器计数器
#define T0PR           (*((volatile unsigned long *) 0xE000400C))     //预分频寄存器
#define T0PC           (*((volatile unsigned long *) 0xE0004010))     //预分频计数器
#define T0MCR          (*((volatile unsigned long *) 0xE0004014))     //匹配控制寄存器
#define T0MR0          (*((volatile unsigned long *) 0xE0004018))     //匹配寄存器0
#define T0MR1          (*((volatile unsigned long *) 0xE000401C))     //匹配寄存器1
#define T0MR2          (*((volatile unsigned long *) 0xE0004020))     //匹配寄存器2
#define T0MR3          (*((volatile unsigned long *) 0xE0004024))     //匹配寄存器3
#define T0CCR          (*((volatile unsigned long *) 0xE0004028))     //捕获控制寄存器
#define T0CR0          (*((volatile unsigned long *) 0xE000402C))     //捕获寄存器0
#define T0CR1          (*((volatile unsigned long *) 0xE0004030))     //捕获寄存器1
#define T0CR2          (*((volatile unsigned long *) 0xE0004034))     //捕获寄存器2
#define T0CR3          (*((volatile unsigned long *) 0xE0004038))     //捕获寄存器3
#define T0EMR          (*((volatile unsigned long *) 0xE000403C))     //外部匹配寄存器
/* 定时器的特殊寄存器 Timer 1*/
#define T1IR           (*((volatile unsigned long *) 0xE0008000))     //与定时器0一致
#define T1TCR          (*((volatile unsigned long *) 0xE0008004))
#define T1TC           (*((volatile unsigned long *) 0xE0008008))
#define T1PR           (*((volatile unsigned long *) 0xE000800C))
#define T1PC           (*((volatile unsigned long *) 0xE0008010))
#define T1MCR          (*((volatile unsigned long *) 0xE0008014))
#define T1MR0          (*((volatile unsigned long *) 0xE0008018))
#define T1MR1          (*((volatile unsigned long *) 0xE000801C))
#define T1MR2          (*((volatile unsigned long *) 0xE0008020))
#define T1MR3          (*((volatile unsigned long *) 0xE0008024))
#define T1CCR          (*((volatile unsigned long *) 0xE0008028))
#define T1CR0          (*((volatile unsigned long *) 0xE000802C))
#define T1CR1          (*((volatile unsigned long *) 0xE0008030))
#define T1CR2          (*((volatile unsigned long *) 0xE0008034))
#define T1CR3          (*((volatile unsigned long *) 0xE0008038))
#define T1EMR          (*((volatile unsigned long *) 0xE000803C))
/* 脉宽调制器的特殊寄存器 Pulse Width Modulator (PWM)*/
#define PWMIR          (*((volatile unsigned long *) 0xE0014000))     //PWM 中断寄存器RW
#define PWMTCR         (*((volatile unsigned long *) 0xE0014004))     //PWM 定时器控制寄存器RW
#define PWMTC          (*((volatile unsigned long *) 0xE0014008))     //PWM 定时器计数器RW
#define PWMPR          (*((volatile unsigned long *) 0xE001400C))     //PWM 预分频寄存器RW
#define PWMPC          (*((volatile unsigned long *) 0xE0014010))     //PWM 预分频计数器RW
#define PWMMCR         (*((volatile unsigned long *) 0xE0014014))     //PWM 匹配控制寄存器RW
#define PWMMR0         (*((volatile unsigned long *) 0xE0014018))     //PWM 匹配寄存器0RW
#define PWMMR1         (*((volatile unsigned long *) 0xE001401C))     //PWM 匹配寄存器1RW
#define PWMMR2         (*((volatile unsigned long *) 0xE0014020))     //PWM 匹配寄存器2RW
#define PWMMR3         (*((volatile unsigned long *) 0xE0014024))     //PWM 匹配寄存器3RW
#define PWMMR4         (*((volatile unsigned long *) 0xE0014040))     //PWM 匹配寄存器4RW
#define PWMMR5         (*((volatile unsigned long *) 0xE0014044))     //PWM 匹配寄存器5RW
#define PWMMR6         (*((volatile unsigned long *) 0xE0014048))     //PWM 匹配寄存器6RW
#define PWMPCR         (*((volatile unsigned long *) 0xE001404C))     //PWM 控制寄存器（使能、单双边沿）RW
#define PWMLER         (*((volatile unsigned long *) 0xE0014050))     //PWM 锁存使能寄存器（锁存新值）RW
```

```c
/* A/D转换器 A/D CONVERTER */   /* Not used for lpc210x  lpc210x除外*/
#define ADCR            (*((volatile unsigned long *) 0xE0034000))     // A/D 控制寄存器
#define ADDR            (*((volatile unsigned long *) 0xE0034004))     // A/D 数据寄存器
/* 实时时钟的特殊寄存器Real Time Clock */
#define ILR             (*((volatile unsigned char *) 0xE0024000))     // 中断位置寄存器RW
#define CTC             (*((volatile unsigned short*) 0xE0024004))     // 时钟节拍计数器RO
#define CCR             (*((volatile unsigned char *) 0xE0024008))     // 时钟控制寄存器（使能、复位、测试）RW
#define CIIR            (*((volatile unsigned char *) 0xE002400C))     // 实时时钟递增中断寄存器RW
#define AMR             (*((volatile unsigned char *) 0xE0024010))     // 实时时钟报警屏蔽寄存器RW
#define CTIME0          (*((volatile unsigned long *) 0xE0024014))     // 完整时间寄存器0（星期、小时、分、秒）RO
#define CTIME1          (*((volatile unsigned long *) 0xE0024018))     // 完整时间寄存器1（年、月、日期[月]）RO
#define CTIME2          (*((volatile unsigned long *) 0xE002401C))     // 完整时间寄存器2（日期[年]）RO
#define SEC             (*((volatile unsigned char *) 0xE0024020))     // 设置秒RW
#define MIN             (*((volatile unsigned char *) 0xE0024024))     // 设置分RW
#define HOUR            (*((volatile unsigned char *) 0xE0024028))     // 设置小时RW
#define DOM             (*((volatile unsigned char *) 0xE002402C))     // 设置日期（月）RW
#define DOW             (*((volatile unsigned char *) 0xE0024030))     // 设置星期RW
#define DOY             (*((volatile unsigned short*) 0xE0024034))     // 设置日期（年）RW
#define MONTH           (*((volatile unsigned char *) 0xE0024038))     // 设置月RW
#define YEAR            (*((volatile unsigned short*) 0xE002403C))     // 设置年RW
#define ALSEC           (*((volatile unsigned char *) 0xE0024060))     // 秒报警值RW
#define ALMIN           (*((volatile unsigned char *) 0xE0024064))     // 分报警值RW
#define ALHOUR          (*((volatile unsigned char *) 0xE0024068))     // 小时报警值RW
#define ALDOM           (*((volatile unsigned char *) 0xE002406C))     // 日期（月）报警值RW
#define ALDOW           (*((volatile unsigned char *) 0xE0024070))     // 星期报警值RW
#define ALDOY           (*((volatile unsigned short*) 0xE0024074))     // 日期（年）报警值RW
#define ALMON           (*((volatile unsigned char *) 0xE0024078))     // 月报警值RW
#define ALYEAR          (*((volatile unsigned short*) 0xE002407C))     // 年报警值RW
#define PREINT          (*((volatile unsigned short*) 0xE0024080))     // 预分频值，整数部分RW
#define PREFRAC         (*((volatile unsigned short*) 0xE0024084))     // 预分频值，小数部分RW
/* 看门狗的特殊寄存器 Watchdog */
#define WDMOD           (*((volatile unsigned char *) 0xE0000000))     // 看门狗模式寄存器（模式、状态）R/W
#define WDTC            (*((volatile unsigned long *) 0xE0000004))     // 看门狗定时器常数寄存器（超时值）R/W
#define WDFEED          (*((volatile unsigned char *) 0xE0000008))     // 看门狗喂狗寄存器（特定喂狗时序）WT
#define WDTV            (*((volatile unsigned long *) 0xE000000C))     // 看门狗定时器值寄存器（反映当前值）RO
/* 定义固件函数 Define firmware Functions */
#define rm_init_entry()             ((void (*)())(0x7fffff91))()
#define rm_undef_handler()          ((void (*)())(0x7fffffa0))()
#define rm_prefetchabort_handler()  ((void (*)())(0x7fffffb0))()
#define rm_dataabort_handler()      ((void (*)())(0x7fffffc0))()
#define rm_irqhandler()             ((void (*)())(0x7fffffd0))()
#define rm_irqhandler2()            ((void (*)())(0x7fffffe0))()
#define iap_entry(a, b)             ((void (*)())(0x7ffffff1))(a, b)

/**********************************************************************************************************
```

这是一个例子

```
/*******************************************************************************
* 文件名：main.c
* 功  能：使用定时器实现秒定时,控制蜂鸣器蜂鸣。(查询方式)
* 说  明：JP4跳线短接，JP7跳线断开。
*******************************************************************************/
#include  "config.h"

#define    BEEPCON   1<<7                              // P0.7引脚控制B1，低电平蜂鸣

/*******************************************************************************
* 名  称：Time0Init()
* 功  能：初始化定时器，定时时间为S。
* 入口参数：无
* 出口参数：无
*******************************************************************************/
void  Time0Init(void)
{   /* Fcclk = Fosc*4 = 11.0592MHz*4 = 44.2368MHz
       Fpclk = Fcclk/4 = 44.2368MHz/4 = 11.0592MHz
    */
    T0PR = 99;                                      // 设置定时器分频为分频，得Hz
     T0MCR = 0x03;                                  // 匹配通道匹配中断并复位T0TC
     T0MR0 = 110592;                                // 比较值(1S定时值)
     T0TCR = 0x03;                                  // 启动并复位T0TC
     T0TCR = 0x01;
}
/*******************************************************************************
* 名  称：main()
* 功  能：初始化I/O及定时器，然后不断的查询定时器中断标志。当定时时间到达时，取反BEEPCON
*          控制口。
*******************************************************************************/
int  main(void)
{
    PINSEL0 = 0x00000000;                           // 设置管脚连接GPIO
    IO0DIR = BEEPCON;                               // 设置I/O为输出
    Time0Init();                                    // 初始化定时器

    while(1)
    {
        while( (T0IR&0x01) == 0 );                  // 等待定时时间到
        T0IR = 0x01;                                // 清除中断标志
        if( (IO0SET&BEEPCON) == 0 )
        {
            IO0SET = BEEPCON;
        }
        else
        {
            IO0CLR = BEEPCON;
        }
   }
   return(0);
}
/*******************************************************************************
```