# STR71X ARM 启动描述

**罗辉联 2006-9**

## ARM 启动概述

ARM 在上电之后，紧接着就从地址 0 开始取得第一条指令的代码来执行，所以在地址 0 的地方必须在上电的时候就存在可执行的正确代码，也就是 0 的地方应该是 ROM(Flash)区，至少在上电之后的一小段时间内是 ROM 区。

对于 ARM 而言，中断和异常的入口地址是 0~20H，每当有中断或者异常发生的时候，PC 就会跳转到 0~20 之间相应的地址去取程序代码执行。如果中断向量处于 ROM 区，那么中断执行程序的地址只能是固定在其中，而不能被程序动态的来改动；另一方面就是，ROM 通常都会比较慢一些，ARM 需要加入等待周期，所以我们希望中断向量位于 RAM 区。

ARM 的 ARM 提供了一个 REMAP 来解决这一个问题。上电复位的时候 0 地址一般被映射在内部 Flash 物理地址 0x40000000,或者外部存储器接口 CS0 所对应的物理地址 0x60000000,在 STR710 ARM 中，程序上电时逻辑 0 地址对应物理地址的选择是通过硬件配置实现的,内部 RAM 映射到 0x20000000 处，当执行相应的REMAP 命令之后，内部的 RAM 就映射到 0 的位置，如果这个时候中断和异常向量表已经被复制了，那么就能够实现我们所需要的情况：上电复位的一段时间内，ARM 可以在 0 地址取到可执行的有效的指令代码，然后在应用程序执行时，能够以更快的速度响应中断和异常，并且可以通过指令的执行动态地进行改动。

存在于 ROM 中的程序有一个重要作用就是初始化 ARM 本身，并且引导其运行起来，真正进入到应用程序的部分去。所以可以将系统初始化的阶段分为两个部分，第一部分初始化 ARM 本身，保证 ARM 能够正常的运行起来，并能顺利进入 C 程序主函 MAIN.，这部分代码一般用汇编实现，第二部分对 ARM 进行全面的低级初始化，一般用 C 语言实现，执行时放在 main 函数的入口处。下面分别介绍两个部分的初始化过程。

# 一、 ARM 基本初始化过程

本部分的内容定义在汇编文件 bootloader.s 中，启动时首先执行。

## 1. 定义 ENTRY POINT

定义程序入口点和代码类型。

## 2. 设置中断\异常向量

是上电瞬间在 REMAP 之前所执行的中断向量，这里假设通过硬件配置是被连接到内部 Flash 物理地址 0x40000000 的地方，但是在 REMAP 之前，是地址 0 的地方。

## 3. 等待 OSC 稳定(在初始化的第一部分通常系统时钟使用默认值,所以需保证 OSC 输出稳定的时钟信号)

等待 OSC 稳定通常是执行多个 NOP 指令实现等待操作,这段代码放在复位异常代码段的开始处.

## 4. 初始化数据访问终止模式，未定义指令终止模式，管理模式堆栈指针

在 OSC 稳定后首先初始化数据访问终止模式，未定义指令终止模式堆栈指针，目的是为对在下面的初始化工程中出现数据访问终止模式,未定义指令终止异常时能够及时得到处理。这两种模式的堆栈指针初始化接着是初始化管理模式堆栈指针，并在此时使 ARM 保持着管理模式,因为接下来的 ARM 本身初始化是运行在除 7 种异常模式之外的模式，而系统一旦进入系统模式和用户模式将无法通过软件在这两种模式下进行 ARM 模式的切换，从而就不能对 ARM 的 FIQ,IRQ 异常的堆栈指针进行初始化。而ARM 处于管理模式时却能实现这个功能。

## 5. 复位并允许所有的外部设备

由于 STR710 的内部外设与 ARM 核通过 APB 桥相联系，为了使内部外设正常工作，通常在系统初始化阶段需要对 APB 桥上的外设复位。具体的操作方法是先禁止挂接在 APB 桥上所有外设的时钟源，然后复位外设，之后允许所有外设的时钟源。

## 6. 初始化存储系统,和 GPIO

由于 STR710 的所有内部外设，片选和部分地址线(A20, A21, A22, A23)都与 GPIO 复用。而在某些应用系统中不是所有的外设，片选，地址线都会被使用，因此这些不被使用的外设或 GPIO 需要有个初始值。所以我们建议在第一部分初始化阶段将所有的 GPIO 初始化为一种工作模式，通常为推拉输出模式，并使 PD 寄存器的值为零，而在第二部分的初始化阶段根据具体的应用进行 GPIO 模式设置,和赋初值。上面提到片选也与 GPIO 复用，如果你系统的 CODE 区为外部 FLASH,内存区为外部 RAM，那么还必须对相应的片选进行初始化，初始化的内容包括：允许与禁止，数据宽度，读写数据时插入等待周期的个数。本例子为调试工程，CODE 和 RAM 区都为外部 RAM 区，所以只是对连接外部 RAM 的片选 CS1 进行初始化。

## 7. 初始化中断控制器，并禁止所有的中断

为了防止在初始化的过程中产生中断，所以必须在系统刚开始运行后不久就初始化

中断控制器，包括：禁止 IRQ 和 FIQ 中断、禁止 IRQ 中断的所有通道 、清除 IRQ 中断各 Pending 位、禁止 FIQ 通道中断，同时清除 FIQ 中断通道 Pending 位、初始化所有 IRQ 通道中断的优先级为零。

### 8. 初始化 FIQ,IRQ,USR 模式堆栈指针

通常中断的堆栈放在内部 RAM 区，响应速度更快；用户应用程序的堆栈区放在外部 RAM 区的最后面，如果你的系统完全使用内部 RAM 就足够了，那么用户和中断堆栈可以都放在内部 RAM 的底部。

### 9. 初始化 C 程序用到的存储区

程序编译连接完成后，CODE 和程序数据(全局变量)是被一同下载到 ROM(Flash)同时存储在其中，当程序运行后所需要的数据(全局变量)是在系统初始化阶段并在进入 C 环境之前将其从 ROM 复制到 RAM(内部或者外部 RAM，根据你需要定制)。在需要复制的数据包括初始化为零的数据(ZI)、未初始化和初始化不为零的数据(RW), 先复制 RW 的数据到 RAM, 然后才复制 ZI 数据到 RAM。

在 ADS 中一般有两种方法来初始化 C 程序用到的存储区，分别介绍如下：

(1) 用户代码初始化 C 程序用到的存储区

对于 ADS 的连接器在连接的时候会生成一些符合分别表示 CODE, RW 数据,ZI 数据开始和结束的区域，所以我们可根据这些连接器生成的符号方面的进行 RW,ZI 数据的复制操作，这部分代码通常放位于进入 main 函数之前。

（2）由编译器自动初始化 C 程序用到的存储区

### 10. 进入 C 程序

在前面的初始化工程中除了初始化异常的堆栈空间时 ARM 运行在相应的异常模式，其他时候均运行于管理模式(SVC)，当程序进入 C 环境后 ARM 是运行在用户(或系统)模式下,所以在真正进入 C 程序之前还应该把 ARM 的模式切换的系统模式或者用户模式，一般是在此处切换到系统模式，并允许 ARM 核的 FIQ,IRQ 异常

注：因为系统一旦进入 C 程序，是不可能返回的，所以此处用 B 分支指令而不用 BL。

## 二、ARM 系统初始化过程

此部分代码全部 C 语言实现，放在 main 函数中执行。

在进入实际的应用之前需要对 ARM 的所有外设进行系统的初始化，使外设恢复为默认值，等到后面具体使用用某个外设时在做相应的配置。如果你为了保证异常响应的数据还需要将异常向量表复制到 RAM（内部或外部）,同时执行存储器的重映射指令。该部分的初始化代码在 **main** 函数的入口处被调用。

### 1. 异常向量拷贝到 RAM，重映射存储器

为了保证异常(中断)响应速度，通常在程序启动后将异常向量复制到 RAM，同时将 RAM 的起始地址映射成 0 地址，当异常出现时 ARM 从该处获得异常向量，并处理之。

### 2. 禁止看门狗

看门狗在防止程序乱飞，进入无限死循环时起着重要作用，有些应用可能不需要看门狗功能，或者某些应用使用外部看门狗，在这个时候内部看门狗必须禁止。所以我们建议在这部分的初始化过程中将看门狗禁止，当应用要选择使用内部看门狗时，到后面的应用中去允许内部看门狗功能

### 3. ARM 时钟系统初始化

时钟时 ARM 运行的基础，在第一部分的初始化过程中，ARM 运行的时钟频率采用了外部 OSC 的默认频率，但在实际的应用中 ARM 核, APB 外设可能使用不同的时钟频率，而 STR710 ARM 提供功能强大的时钟控制模块，可为 ARM，APB 等外设提供多种可选的时钟频率，具体通过时钟模块的倍频与分频器的组合使用而产生不同频率的时钟源。我们也建议在该处初始化时将应用系统所需配置的时钟(USB 设备除外)都在此处配置完成。

### 4. 存储系统重新初始化

在一些应用系统中除了外扩 Flash,RAM 挂接在外部存储器接口上外，可能还有其他的外设挂接在外部存储器接口上，不同外设置的操作时序是不相同的，所以在使用这些外设之前必须初始化连接这些外设的存储器接口。我们建议应用程序中所使用的存储器接口都在这里初始化。

### 5. APB 外设初始化

APB 外设初始化主要是将 APB 上相关的外设配置为复位的默认值，当应用程序中需要使用使用某个模块时，在根据应用配置成需要的值。

### 6. 允许 IRQ,FIQ 全局中断

## 三、 ARM 启动过程实例---ARM 基本初始化

**bootloader.s 代码实例**

```
/*****************************************Copyright(c)*****************************
**--------------文件信息--------------------------------------------------------------------------------
** 文 件 名: init.c
** 创建日期: 2006 年 5 月 2 日
** 描       述: str710 启动代码
**--------------历史版本--------------------------------------------------------------------------------
** 日       期: 2006 年 5 月 2 日
** 描       述: 原始版本
**------------------------------------------------------------------------------------------------------
********************************************************************************/
PRESERVE8
;处理器各种模式标志定义
SVC_MODE       EQU     0x13                    管理模式
USR_MODE       EQU     0x10                    用户模式
SYS_MODE       EQU     0x1F                    系统模式
IRQ_MODE       EQU     0x12                    中断模式
FIQ_MODE       EQU     0x11                    快速中断模式
ABT_MODE     EQU     0x17                    中止异常模式
UNF_MODE     EQU     0x1B                    未定义指令异常模式
MODE_MASK      EQU     0x1F                    屏蔽任何模式
I_BIT   EQU  0x80                              禁止 IRQ 中断
F_BIT   EQU     0x40                           禁止 FIQ 中断
NO_INT         EQU     0xC0                    中断锁定配置
;各种模式堆栈大小定义,注意：SIZE 的大小必须是 4 的倍数
SVC_STACK_SIZE  EQU     256                    定义系统堆栈大小
SYS_STACK_SIZE  EQU     1024                   定义系统堆栈大小
IRQ_STACK_SIZE  EQU     512
FIQ_STACK_SIZE  EQU     256             .
ABT_STACK_SIZE  EQU     (1*4)           1              个字
UNF_STACK_SIZE  EQU     (1*4)           1              个字

IRQ_FIQ_SYS_STACK_SIZE EQU   IRQ_STACK_SIZE + FIQ_STACK_SIZE +
SYS_STACK_SIZE
```

```
ABT_UND_SVC_STACK_SIZE  EQU   ABT_STACK_SIZE + UNF_STACK_SIZE +
SVC_STACK_SIZE
;重映射逻辑变量定义
; GBLL   REMAP
;引入的外部标号或函数
 IMPORT  Undefined_Handler
 IMPORT  SWI_Handler
 IMPORT  PrefetchAbort_Handler
 IMPORT DataAbort_Handler

 IMPORT  IRQ_Handler                                  通用中断处理函数
 IMPORT  FIQ_Handler                                  快速中断异常处理程序

 IMPORT ChipResetInit      STR710                          复位初始化
 IMPORT __main       C                                  语言入口函数

;给外部使用的标号或者函数
 EXPORT INT_Initialize
   EXPORT __rt_div0
   EXPORT __user_initial_stackheap                    库函数初始化堆和栈

;系统启动部分代码段
 CODE32
   AREA    start,CODE,READONLY

   ENTRY                                              程序入口点
********************************************************************************
;                                异常向量表
********************************************************************************
 EXPORT  InitVectors
InitVectors
      LDR     PC, Reset_Addr
      LDR     PC, Undefined_Addr
      LDR     PC, SWI_Addr
      LDR     PC, Prefetch_Addr
```

```
    LDR    PC, Abort_Addr
    NOP                                          保留向量
    LDR    PC, IRQ_Addr
    LDR    PC, FIQ_Addr


Reset_Addr     DCD    INT_Initialize
Undefined_Addr  DCD    Undefined_Exception
SWI_Addr       DCD    SWI_Exception
Prefetch_Addr   DCD    PrefetchAbort_Exception
Abort_Addr     DCD    DataAbort_Exception
          DCD    0                               保留向量
IRQ_Addr       DCD    IRQ_Exception
FIQ_Addr       DCD    FIQ_Exception

;/*****************************************************************************
;**                                    异常处理实例
;**    根据 ATPCS的规则: R0-R3 通常用来传递参数和保存结果
;**             R4-R11                通常用来保存局部变量
;**             R12                   通常用作子程序内部调用的 scratch 寄存器
;**                                   所以当异常出现时 R0-R12,LR 等寄存器均需要入栈进行保护
;**    异常 Handler 说明: 各个异常相对应的实例定义在 irq.c 中,实际处理程序在其中添加
;**  IRQ   中断嵌套说明: 在本例子中 IRQ 中断允许嵌套，但缺省状态未为中断嵌套进行处
;*****************************************************************************/
;未定义指令
Undefined_Exception
          STMFD   SP!, {R0-R11, LR}
          MRS     R1, SPSR
          STMFD   SP!,{R1}
          BL      Undefined_Handler             来自 irq.c 文件
          MSR     SPSR_cxsf,R1                  恢复中断前的 CPSR
          LDMFD   SP!,{R0-R11,PC}^              从未定义指令异常返回
;软件中断
SWI_Exception
          STMFD   SP!, {R0-R11, LR}
```

```
        MRS     R1, SPSR
        STMFD   SP!,{R1}
        BL      SWI_Handler                 来自 irq.c 文件
        MSR     SPSR_cxsf,R1                恢复中断前的 CPSR
        LDMFD   SP!,{R0-R11,PC}^            从软件中断异常返回
;取指令中止
PrefetchAbort_Exception
        SUB     LR,LR,#4
        STMFD   SP!, {R0-R11, LR}
        MRS     R1, SPSR
        STMFD   SP!,{R1}
        BL      PrefetchAbort_Handler       来自 irq.c 文件
        MSR     SPSR_cxsf,R1                恢复中断前的 CPSR
        LDMFD   SP!,{R0-R11,PC}^            从取指令中止异常返回
;取数据中止
DataAbort_Exception
        SUB     LR,LR,#8
        STMFD   SP!, {R0-R11, LR}
        MRS     R1, SPSR
        STMFD   SP!,{R1}
        BL      DataAbort_Handler           来自 irq.c 文件
        LDMFD   SP!,{R1}
        MSR     SPSR_cxsf,R1                恢复中断前的 CPSR
        LDMFD   SP!,{R0-R11,PC}^            从取数据中止异常返回
;/*****************************************************************************
;**  函数名称: FIQ_Handler
;**  功能描述:  快速中断入口,即 FIQ 异常处理实例
;**  参       数: None
;**  返  回  值: None
;**  作       者:  罗辉联
;**  日       期: 2006 年 5 月 18 日
;**-----------------------------------------------------------------------------------------------
;***************************************************************************** /

FIQ_Exception
```

```
        SUB    LR,LR,#4                保存实际的返回地址
        STMFD  SP!,{R0-R7, LR}         保存 R0-R7  和 LR  到 FIQ  堆栈,R8~R14
                                       FIQ 模式为专用积存器不用保存到堆栈
        MRS    R1, SPSR
        STMFD  SP!,{R1}                保存 FIQ 出现前 CPSR 的值到堆栈
        BL     FIQ_Handler             响应 FIQ 异常,来自 irq.c 文件
        LDMFD  SP!,{R1}
        MSR    SPSR_cxsf,R1            恢复中断前的 CPSR
        LDMFD  SP!,{R0-R7,PC}^         从 IRQ 返回

;/*****************************************************************************
;** 函数名称: IRQ_Handler
;** 功能描述: IRQ 中断入口,即 IRQ 异常处理实例
;** 参      数: None
;** 返 回 值: None
;** 作      者: 罗辉联
;** 日      期: 2006 年 5 月 18 日
;**-----------------------------------------------------------------------------------------
;***************************************************************************** /
IRQ_Exception
        SUB    LR,LR,#4                保存实际的返回地址
        STMFD  SP!,{R0-R11,LR}     R0-R3       通常用来传递参数和保存结果,R4-R11
                                       通常保存局部变量
        MRS    R1, SPSR
        STMFD  SP!,{R1}                保存 IRQ 出现前 CPSR 的值到堆栈
        BL   IRQ_Handler               响应 IRQ 异常,来自 irq.c 文件
        LDMFD  SP!,{R1}
        MSR    SPSR_cxsf,R1                恢复中断前的 CPSR
        LDMFD  SP!,{R0-R11,PC}^            从 IRQ 返回
;/*****************************************************************************
;** 函数名称: INT_Initialize
;** 功能描述: 复位入口
;** 参      数: None
;** 返 回 值: None
;** 作      者: 罗辉联
```

```
;**  日       期: 2006 年 5 月 18 日
;**--------------------------------------------------------------------------------------------------
;*********************************************************************************** /
INT_Initialize
                    LDR     PC, =NextInst
;等待 OSC 稳定
NextInst
          NOP                                   系统启动阶段采用默认的晶振频率
          NOP
          NOP
          NOP
          NOP
          NOP                   应用程序运行后需要更改时钟频率，可在 C 代码中重新设置
          NOP
          NOP
          NOP
;保证一开始处理器处于管理模式,且禁止 FIQ,IRQ 中断
          MRS     a1,CPSR                       获得当前的 CPSR
          BIC     a1,a1,#MODE_MASK               清除当前的工作模式位
          ORR     a1,a1,#SVC_MODE                设置管理模式位
          ORR     a1,a1,#NO_INT                 保证 IRQ/FIQ 处于禁止状态
          MSR     CPSR_cxsf,a1                  设置新的 CPSR
;获得 ABT,UND,SVC 模式堆栈栈低位置
          LDR     a1,[pc, #ABT_UND_SVC_STACK_SEGMENT-.-8]
;建立 ABORT 模式下的数据栈
          MOV     a2,#ABT_STACK_SIZE            获得 Abort 模式堆栈大小
          SUB     a2,a2,#4                      减去一个字，获得 ABT_UND_SVC
                                                 堆栈的开始地址
          ADD  a3,a1,a2                         设置 Abort 模式堆栈区间
          BIC     a3,a3,#3                      保证字对齐
          MRS     a1,CPSR                       获得当前 CPSR
          BIC     a1,a1,#MODE_MASK              清除当前的工作模式位
          ORR     a1,a1,#ABT_MODE               设置 Abort 模式位
          MSR     CPSR_cxsf,a1                  写入 Abort 模式位
          MOV     sp,a3                         设置 Abort 模式堆栈
```

```
;建立 UNDEF 模式下的数据栈
        MOV    a2,#UNF_STACK_SIZE          获得 Undefined 模式堆栈大小
        ADD  a3,a3,a2                      设置 Undefined 模式堆栈区间
        BIC    a3,a3,#3                    保证字对齐
        MRS    a1,CPSR              ;
        BIC    a1,a1,#MODE_MASK            清除当前的工作模式位
        ORR    a1,a1,#UNF_MODE             设置 Undefined 模式位
        MSR    CPSR_cxsf,a1         ;
        MOV    sp,a3                       设置 Undefined 模式堆栈
;建立管理模式下的数据栈,并回到管理模式
        MOV    a2,#SVC_STACK_SIZE          获得 SVC 模式堆栈大小
        ADD  a3,a3,a2                      设置 SVC 模式堆栈区间
        BIC    a3,a3,#3                    保证字对齐
        MRS    a1,CPSR              ;
        BIC    a1,a1,#MODE_MASK            清除当前的工作模式位
        ORR    a1,a1,#SVC_MODE      ;
        MSR    CPSR_cxsf,a1                回到管理模式,也为了后面的操作
        MOV    sp,a3                       设置 SVC 模式堆栈

        BL   ChipResetInit                 芯片复位初始化,在 ChipReset.s 中定义
                                           注意重映射与向量拷贝放在 C 代码中执行
;获得 IRQ,FIQ,SYS 等模式堆栈栈低位置
        LDR    a1,[pc, #IRQ_FIQ_SYS_STACK_SEGMENT-.-8]
;建立 FIQ 模式下的数据栈
        MOV    a2,#FIQ_STACK_SIZE          获得 FIQ 模式堆栈大小
        SUB    a2,a2,#4                    减去一个字，获得 IRQ_FIQ_SYS 堆
                                           栈的开始地址
        ADD  a3,a1,a2                      设置 FIQ 模式堆栈区间
        BIC    a3,a3,#3                    保证字对齐
        MRS    a1,CPSR              ;
        BIC    a1,a1,#MODE_MASK            清除当前的工作模式位
        ORR    a1,a1,#FIQ_MODE             设置 FIQ 模式位
        MSR    CPSR_cxsf,a1         ;
        MOV    sp,a3                       设置 FIQ 模式堆栈
;建立 IRQ 模式下的数据栈
```

```
        MOV    a2,#IRQ_STACK_SIZE              获得 IRQ 模式堆栈大小
        ADD  a3,a3,a2                          设置 IRQ 模式堆栈区间
        BIC    a3,a3,#3                        保证字对齐
        MRS    a1,CPSR                 ;
        BIC    a1,a1,#MODE_MASK                清除当前的工作模式位
        ORR    a1,a1,#IRQ_MODE                 设置 IRQ 模式位
        MSR    CPSR_cxsf,a1            ;
        MOV    sp,a3                           设置 IRQ 模式堆栈
;建立 SYS 模式下的数据栈
        MOV    a2,#SYS_STACK_SIZE              获得 SYS 模式堆栈大小
        ADD  a3,a3,a2                          设置 SYS 模式堆栈区间
        BIC    a3,a3,#3                        保证字对齐
        MRS    a1,CPSR                 ;
        BIC    a1,a1,#MODE_MASK                清除当前的工作模式位
        ORR    a1,a1,#SYS_MODE                 设置 SYS 模式位
        MSR    CPSR_cxsf,a1            ;
        MOV    sp,a3                   设置 SYS 模式堆栈,USR 与 SYS 共用堆栈空间
        MSR    CPSR_cxsf, #SYS_MODE                  允许 IRQ 和 FIQ 中断
        B      __main     ;
```

//进入 main 后程序不会返回，所以用 B 而不是 BL 指令执行__main 本工程的启动代码使用编译器生成的初始化函数对所有应用程序的数据区进行设置,包含 RW，ZI 段的数据,C 运行库和额外的堆区也在此处初始化，一旦运行环境设置完成，初始化代码就调用应用程序的"主"如口点,而__main 就是编译生成代码的主入口点。

```
;/*******************************************************************************
;**  函数名称: __user_initial_stackheap
;**  功能描述: 库函数初始化堆和栈，不能删除
;**  参       数: 参考库函数手册
;**  返  回  值: 参考库函数手册
;**  作       者: 罗辉联
;**  日       期: 2006 年 5 月 18 日
;**-------------------------------------------------------------------------------------------------------
;******************************************************************************* /
__user_initial_stackheap
        LDR   R0,BOTTOM_OF_HEAP
```

```
        MOV   PC,LR

;/*****************************************************************************
;**  函数名称: __rt_div0
;**  功能描述:  整数除法除数为 0 错误处理函数
;**  参       数:  参考库函数手册
;**  返  回  值: None
;**  作       者:  罗辉联
;**  日       期: 2006 年 5 月 18 日
;**-----------------------------------------------------------------------------------------
;***************************************************************************** /
__rt_div0
            B      __rt_div0

ABT_UND_SVC_STACK_SEGMENT  DCD  abt_und_svc_stack
IRQ_FIQ_SYS_STACK_SEGMENT  DCD  irq_fiq_sys_stack
BOTTOM_OF_HEAP    DCD  bottom_of_heap
*****************************************************************************
;                                  分配堆空间
*****************************************************************************
AREA    lib_function_heap, DATA, NOINIT, ALIGN=2

EXPORT bottom_of_heap
bottom_of_heap    SPACE   256                              库函数的堆空间
*****************************************************************************
;                                  分配堆栈空间
*****************************************************************************
AREA    ExceptionStacks, DATA, NOINIT, ALIGN=2

EXPORT irq_fiq_sys_stack
EXPORT abt_und_svc_stack
irq_fiq_sys_stack      SPACE   IRQ_FIQ_SYS_STACK_SIZE
abt_und_svc_stack     SPACE   ABT_UND_SVC_STACK_SIZE
        LTORG
    END
```