北京世纪联信科技有限公司
Beijing Century Connected-Information Science&Technology Ltd.

AT91 Assembler Code Startup Sequence for C Code Applications Software Based on the AT91SAM7S64 Evaluation Board

（中文版）

北京世纪联信科技有限公司
中国　北京　海淀区知春路碧兴园 A 座 2305 号
邮编：100088
电话：（010）51735791/2
传真：（010）51735793
网站：www.91arm.cn

基于 AT91SAM7S64 评估版 C 代码应用程序软件的 AT91 汇编代码启动过程

1. 介绍

出于模块化和可移植性的考虑，绝大多数基于 ARM® 微处理器的 AT91SAM7S 的应用程序代码都是用 C 语言编写的，但是，由于其启动过程需要初始化 ARM 处理器模式和一些对寄存器结构和存储器映射处理器依赖性很强的重要外设，因而其中的 C 启动顺序是用汇编语言编写的。

本应用笔记分析了 AT91SAM7S C-startup 启动时序的例子，基于 AT91SAM7S64 的 C-startup 过程，编译工具为 IAR 4.11A 开发工具。更多的有关 C-startup 时序的例子都可以在 AT91 软件包中找到。

C-startup 过程在上电复位后被激活。

2. C-Startup 过程

复位后紧接着，处理器开始从地址 0x0 处取指令，因此在这个位置必须要有一些可执行代码。在 AT91SAM7S 嵌入式系统中，要求片内 Flash 在地址 0x0 处，或者至少开始的那一部分在。

最简单的情况就是将片内 Flash 内的应用程序在存储器映射中将其映射到地址 0x0 处，然后这个程序便可以在执行地址 0x0 处复位向量的第一条指令时跳转到其真正的入口点。

所有为基于 ARM 的 AT91SAM7S 系统所编写的应用程序都存储于片内 Flash 中，在复位时执行。

在编写嵌入式操作系统时，或者编写无操作系统的、在复位时执行的嵌入式应用程序时，要考虑到下面一些东西：

- 在片内 Flash 中复位入口点
- 初始化执行环境，比如异常向量、堆栈和 I/O 等
- 初始化应用程序
- 例如，将初始化变量的初始值从 Flash 中复制到 RAM 中，并把其他无关变量清零
- 将一个片内可执行映像与 RAM 中的特殊位置的空间代码和数据联系起来

对一个没有操作系统的片内应用程序来说，Flash 中的代码必须能为程序提供一个自身初始化并开始执行的途径。由于在复位时不会有自动的初始化，所以应用程序的入口点必须在其调用 C 语言代码之前完成初始化。

复位时位于地址 0x0 的初始代码，必须要完成下面一些操作：

- 给出初始代码的默认入口点
- 设置异常向量
- 初始化存储器系统
- 初始化所有必需的 I/O 设备
- 初始化堆栈指针寄存器
- 初始化所有用于中断系统的寄存器
- 使能中断(如果初始代码中用到的话)
- 必要的时候改变处理器模式
- 必要的时候改变处理器状态

在环境初始化完成后，继续初始化应用程序并进入 C 语言代码。

C-startup 文件是上电以后执行的第一个文件，其完成微控制器的初始化，从复位向量一直到调用应用程序的主例行程序。

主程序应当为一个循环，且不会返回。ARM 内核在复位时从地址 0x0 处开始执行指令。对于 AT91SAM7S 嵌入式系统来说，这就意味着系统复位时从片内 Flash 的地址 0x0 处开始。

3. C - Startup 示例

本应用笔记中包含有一个一般的 start-up 文件，其他的都可以在 AT91 软件包里面找到。这里描述的例子是基于 AT91SAM7S64 评估版、IAR V4.11A开发工具下的 C-startup 过程，用于 Flash 存储器调试或 RAM 存储器调试。对于用户不同应用程序的需要，此文件需要作适当的修改。

在 AT91 软件包中的“ 编译 ”子目录里有关于 AT91SAM7S64 的说明，每一个这样的子目录中都包括下面几个文件：

- board.h 文件，用 C 语言定义评估版的组件
- 一个 Cstartup.s79 文件，根据使用的软件开发工具定义评估版的标准启动情况
- 一个 Cstartup_xxx.c 文件，根据使用的软件开发工具定义评估版的标准低级初始化情况

AT91 软件包提供的 C-Startup 文件解释了如何启动 AT91SAM7S 设备以及如何跳转至 C 语言主函数。C-Startup 文件把芯片的特点、评估版的属性以及所需要的调试都考虑在内。

3.1 区域定义

在 ARM 汇编语言的源文件中，程序块的启动是由 PROGRAM 指令标记的，这个指令为链接程序设定程序块。程序块的第二个指令定义了一个名为 RSEG 的程序段，以及定义了 C-startup 和异常代码区域的 ICODE 程序段。

复位时，ARM 内核设置 ARM 模式并取一个 32 位 ARM 指令。CODE32 指令设置后面的所有指令都被读作 32 位 ARM 指令。

汇编指令 ORG 在片内 Flash 的地址 0x0 处生成了一个映像。

```
;------------------------------------------------------------------------
    PROGRAM ?RESET
    RSEG   ICODE:CODE:ROOT(2)
    CODE32  ; Always ARM mode after reset
    org   0
reset:
```

3.2 启动异常向量

异常向量通过地址空间按顺序启动，可跳转到附近的标签或跳转、链接到子程序中。在程序的正常工作流程中，程序计数器增加使能处理器处理由内部或外部来源产生的事件。在正常的工作流程发生转移时，处理器产生异常，这样的例子有：

- 外部产生的中断
- 处理器试图执行一个未定义的指令

之前的处理器的状态存于 SPSR 寄存器中，链接寄存器 (R14) 和堆栈寄存器 (R13) 同样也被硬编码的程序保存。在处理类似异常时，为了使发生异常时正在运行的程序可以在相应异常程序结束后重新恢复运行，在初始化代码中必须启动必需的异常向量。(见 Table 3-1)

Flash 位于地址 0x0 处，异常向量是由一系列跳转的硬编码指令组成，用于处理各种异常。 这些向量都映射到地址 0x0……

Table 3-1. 异常向量

异常	描述
复位	处理器复位引脚拉高时发生，此异常仅作为上电或复位的信号，通过跳转至此复位向量可以实现软件复位。
未定义指令	当处理器或其他协处理器识别出当前运行指令未定义时发生。
软件中断 (SWI)	这是一个用户定义的同步中断指令，其允许运行在用户模式下的程序请求在特权模式下运行一些程序，例如，实时操作系统（RTOS）函数等。
预取中止	当处理器试图执行一个预取自非法地址的指令时发生。
数据中止	当数据传输指令试图在错误的地址装载或存储数据时发生。
IRQ	当处理器外部中断请求引脚有效（低电平）时且 CPSR 寄存器的第 1 位为 0 时发生。
FIQ	当处理器外部快速中断请求引脚有效（低电平）时且 CPSR 寄存器的 F 位为 0 时发生，或者是当一个内部中断被强制重新定为快速中断时发生。

处理器异常处理是由一个向量表控制的，向量表是一个 32 字节的保留区域，通常位于存储器映射的底部，每个向量类型都分配有大小为一个字的空间，还有一个字的保留空间。由于没有足够多的空间容纳除了 FIQ 中断以外其他中断的处理程序的代码，每个异常类型的向量入口都包含了一个跳转指令或者载入 PC 指令以继续执行相应的处理程序。FIQ 异常处理程序可以直接写进异常向量中。

```
;--------------------
;- Exception vectors
;--------------------
       B     InitReset ; 0x00 Reset handler
undefvec:
       B     ndefvec ; 0x04 Undefined Instruction
swivec:
       B     swivec ; 0x08 Software Interrupt
pabtvec:
       B     pabtvec ; 0x0C Prefetch Abort
dabtvec:
       B     dabtvec ; 0x10 Data Abort
rsvdvec:
       B     rsvdvec ; 0x14 reserved
irqvec:
       B     I RQ_Handler_Entry   ; 0x18 IRQ
fiqvec:                          ; FIQ Handling
```

Table 3-2. 异常向量映射

映射地址	异常向量
0x0000 0000	复位
0x0000 0004	未定义指令
0x0000 0008	软件中断 (SWI)
0x0000 000C	预取中止
0x0000 0010	数据中止
0x0000 0014	保留
0x0000 0018	IRQ
0x0000 001C	FIQ

3.3 复位处理程序

从这里起，代码从地址 0x0 处开始执行。

```
;----------------
;- Reset Handler
;---------------- InitReset:
```

3.4 低级初始化

复位后，PLL、片内 Flash 控制器和看门狗定时器都没有配置，而且有些必须在使能中断之前初始化的外围设备也要考虑在内。如果这些外围设备没有在这个时候初始化，它们就很有可能在中断使能的时候产生伪中断。

AT91F_LowLevelInit 函数是在 AT91 软件包中与评估版相关的 C 文件中定义的。汇编 C-startup 把 C 堆栈设置在 RAM 地址的最后，以便于在 C 语言初始化之前调用这个函数。这个函数可以用 Thumb® 指令或 ARM 指令编写，其可以被 BX 切换 ARM 指令调用。

```
;-----------------------
;- Low level init
;-----------------------
;- minimum C initialization
;- call  AT91F_LowLevelInit(void)
       ldr     r13,=__iramend; temporary stack in internal RAM
;--Call Low level init function in ABSOLUTE through the Interworking
       ldr     r0,=AT91F_LowLevelInit
       mov     lr, pc
       bx      r0
```

3.5 AT91F_LowLevelInit 函数

此函数实现非常低级的硬件的初始化，函数可以自身初始化。

- Flash 等待状态和时间设置取决于 PLL 设置和外部振荡器。

复位时，AT91SAM7S 微控制器以慢速时钟为时钟信号从 Flash 默认值启动。

- 禁止看门狗定时器

复位时，AT91SAM7S 微控制器已使能看门狗定时器。

• 设置 PLL

复位时，AT91SAM7S 微控制器以内部慢速时钟 RC 振荡器为时钟信号以使启动系统所需的能耗为最小，这时主时钟是被禁止的。PLL 可以通过设置配置启动，从而通过 PLL 加速启动过程。

• AIC 向量初始化

复位时，先进中断控制器(AIC) 没有配置。AT91F_LowLevelInit 函数通过设置默认中断向量来初始化 AIC。默认的中断处理程序函数在 C-startup 文件中定义。这些函数可以被重写编写。

下面给出的这个函数是用于 AT91SAM7S64 评估版初始化的，特殊指令 “@ ICODE” 链接到 C-startup 程序段区域的目标代码。

```
void AT91F_LowLevelInit(void) @ "ICODE"
{
 int            i;
 AT91PS PMC     pPMC = AT91C BASE PMC;
//* Set Embedded Flash Controller
  AT91C_BASE_MC->MC_FMR= ((AT91C_MC_FMCN)&(50 <<16)) | AT91C_MC_FWS_1FWS;
//* Watchdog Disable
  AT91C_BASE_WDTC->WDTC_WDMR= AT91C_SYSC_WDDIS;
//* Set MCK
// 1 Enabling the Main Oscillator:
  pPMC->PMC_MOR= ((AT91C_CKGR_OSCOUNT & (0x06<<8) | AT91C_CKGR_MOSCEN));
// Startup time
  while(!(pPMC->PMC_SR & AT91C_PMC_MOSCS));
//* Set PLL
  pPMC->PMC_PLLR= ((AT91C_CKGR_DIV & 0x05) |
                          (AT91C_CKGR_PLLCOUNT & (16<<8)) |
                          (AT91C_CKGR_MUL & (25<<16)));
//* Startup time
  while(!(pPMC->PMC_SR & AT91C_PMC_LOCK));
//* select the PLL clock divided by 2
  pPMC->PMC_MCKR= AT91C_PMC_CSS_PLL_CLK | AT91C_PMC_PRES_CLK_2;
//* Set default interrupts handler vectors
  AT91C_BASE_AIC->AIC_SVR[0]= (int) AT91F_Default_FIQ_handler;
  for (i=1;i < 31; i++)
  {
    AT91C_BASE_AIC->AIC_SVR[i]= (int) AT91F_Default_IRQ_handler;
  }
  AT91C_BASE_AIC->AIC_SPU= (int) AT91F_Spurious_handler;
}
```

3.6 初始化 ARM 模式寄存器

中断以及特权堆栈位于 RAM 存储器的顶部，一般来说，中止状态、未定义指令和用户堆栈都不在同一个片内系统中使用。

C-startup 代码初始化堆栈指针寄存器。根据所需的中断和异常的不同，下面这些堆栈指针的一些或全部需要进行初始化：

- 特权堆栈必须要被初始化。
- 如果使用 IRQ 中断的话，IRQ 中断必须被初始化，且一定要在中断被使能之前初始化。
- 标准 AT91SAM7S FIQ 处理程序不使用 FIQ 堆栈，因而不需要初始化 FIQ 堆栈，仅 FIQ 寄存器需要初始化。
- 如果要处理数据和预取中止的话，则必须初始化中止状态堆栈。
- 如果要处理未定义指令的话，则必须初始化未定义指令堆栈。

如果 IRQ 处理程序使用的话，当使用向量时中断堆栈需要 2 字 x 8 优先级 x 4 字节。中断堆栈必须根据中断处理程序的情况做出调整。为节省存储器空间其他堆栈均未被定义。

```
;---------------------
;- Setup each mode
;---------------------
       RSEG   INTRAMEND_REMAP
#define  __iramend SFB(INTRAMEND_REMAP)
       ldr  r0, =__iramend
;- Set up Fast Interrupt Mode and set FIQ Mode Stack
       msr   CPSR_c, #ARM_MODE_FIQ | I_BIT | F_BIT
;- Init the FIQ register
       ldr  r8, =AT91C_BASE_AIC
;- Set up Interrupt Mode and set IRQ Mode Stack
       msr    CPSR_c, #ARM_MODE_IRQ | I_BIT | F_BIT
       mov    r13, r0                     ; Init stack IRQ
       sub    r0, r0, #IRQ_STACK_SIZE
```

3.7 更改处理器模式和使能中断

如果必要的话，初始化代码可以在这个时候把 CPSR 的中断禁止位置 0，使能中断。这里是最早的可以安全使能中断的地方。在这一个阶段处理器仍运行在特权模式下。

```
;------------------------------------------------------------------------
-
;- Setup Application Operating Mode Enable the interrupts and set Stack
;------------------------------------------------------------------------
-
       msr   CPSR_c, #ARM_MODE_SVC
       mov   r13, r0
```

3.8 初始化软件变量并跳转到主函数

下一个任务是进入循环初始化数据存储器，把分配给数据存储和代码的空间写零，这看上去是多余的，但是实际上有下面这两个原因：

1. 在 C 语言中，任何一个未初始化的变量的初始值都应当为零。
2. 这使程序的行为可重复，即便是在部分变量没有被明确的初始化的情况下。
   - 初始化变量的初值表（C 语言下）复制到 RAM 中变量所在的位置。
   - 链接程序使变量的初始值的顺序和 RAM 中的保持一致，对变量初始化来说，块复制就已经足够了。
   - 链接程序把 RAM 源代码段存于片内 Flash 区域中。

所有初始化变量的初始值和 RAM 代码必须从 Flash 复制到 RAM 中，所有其他的变量必须初始化为 0。“__Segment_init” 函数复制 RAM 区域中相应的片内 Flash 代码并初始化数据段。C 初始化是由 IAR 启动函数 __segment _init 完成的，这个函数包含在 C-IAR 库中，可以在 Thumb 或 ARM 指令集中使用。

```
;-------------------------------------------------------
;- Branch on C initialization code (with interworking)
;-------------------------------------------------------
   EXTERN__segment_init
; Initialize segments.
; __segment_init is assumed to use
        ldr  r0,=__segment_init
        mov  lr, pc
       bx       r0
```

当编译器编译一个调用主函数 main() 的函数时，它便产生一个到主函数入口标签处的 PUBLIC 引用。主函数 main() 应当为一个循环，而且不会返回。

```
;-------------------------------------------------------
;- Branch on C code Main function (with interworking)
;-------------------------------------------------------
       EXTERN main
       ldr   r0,=main
       bx    r0
```