Study ARM Step by Step

Study ARM Step by Step

Nick.wang 编著

自序

本文从学习者的角度出发,分别描述了下面几部分内容:
ARM 编程的基本知识,BOOT 代码流程和功能分析,OS 中断程序的编写举例和 BOOT 代码的流程图,希望这些内容能为初学 ARM 的朋友拨开迷雾,以最快的速度和最短的时间走进嵌入世界的大们.

由于编写时间比较急(呵呵,因为还要工作养家糊口),所以错误不可避免,希望各位朋友能指出错误和缺陷.

目录

第一章:ARM ABC 1

- THE ARM PROCESSOR 1
  - 缩写..... 1
  - 处理器模式及对应的寄存器 1
  - ARM寄存器总结 4
- ARM INSTRUCTIONS 5
  - 指令集概述 5
  - 指令的条件执行 7
  - 程序分支..... 8
  - Data Movement Memory Reference Instructions..... 9
- EXAMPLES.....10
  - 向量乘..... 10
  - 字符串比较 11
  - 子程序调用 12

第二章:引导代码分析 13

- 前言.....14
- 概述.....14
  - 与BOOT相关硬件:FLASH ROM 14
  - BOOT的主要功能 17
- 执行流程及代码分析.....22
  - 参数初始化 22
  - 中断..... 24
  - 初始化硬件 36
  - 跳转到C语言程序，开始第二阶段的初始化和系统引导. 47
  - 初始化堆栈 47

第三章:中断服务程序编写 52
- 必需的变量定义.....52
  - 服务程序地址 53
  - I/O端口 53
  - INTERRUPT 控制寄存器 54
  - EINT4567 的Pending 位..... 54
- 变量解释.....54
- 中断服务程序的实现.....58
  - 定义中断服务程序 58
  - 主程序..... 59
  - 中断服务子程序中关键的变量类型 61
  - 断服务程序运行流程图 70

第四章:ＢＯＯＴ流程图 72

附录:BOOT程序源代码..... 79

第 1 章

第一章:ARM 编程 ABC

➢ The ARM Processor

✧缩写

- ✓ ARM:Advanced RISC Machines
- ✓ RISC:Reduced Instruction Set Computer(精简指令集)

✧处理器模式及对应的寄存器

ARM支持下面6种处理器模式,在正常情况下,程序都在用户模式下执行,当软件异常或硬件中断发生时,进入相应的处理器模式.

(1):用户(usr):正常程序执行模式

寄存器:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13,r14,r15,
CPSR

(2):FIQ(fiq):高速数据传送或通道处理

寄存器:

r0,r1,r2,r3,r4,r5,r6,r7,r8_frq,r9_frq,r10_frq,r11_frq
,r12_frq,r13_frq,r14_frq,r15,CPSR

(3):IRQ(irq):通用中断处理

寄存器:

r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13_irq,r14_irq
,r15,CPSR

(4):管理(svc):操作系统保护模式

寄存器:

r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13_svc,r
14_svc,r15,CPSR

(5):中止(abt):存储器保护

寄存器:

r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13_abt,
r14_abt,r15,CPSR

(6):未定义(und):未定义指令

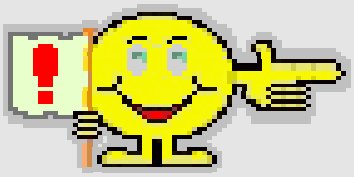
寄存器:

r0,r1,r2,r3,r4,r5,r6,r7,r8,r9,r10,r11,r12,r13_und

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

,r14_und,r15,CPSR

在程序正常执行时,如果发生外部普通中断IRQ,就会进入IRQ处理器模式,这时处理器使用r13_irq和r14_irq,而不是r13和r14.



注释

R15: 程序计数器. program counter 她指向的是下一条要执行的指令而不是正在执行的指令.

R14: 当执行带链接分支的指令 BL 时,得到 R15 的副本.

R13: 作堆栈指针(SP),通常在 BOOT 程序里把 r13 初始化成指向为异常模式分配的堆栈.

异常处理程序将用到的寄存器的值保存到堆栈里,返回时,重新将这些值加载到寄存器.

举例: 未定义模式堆栈初始化

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
UNDEFMODE       DEFINE    0x1b (00011011)

MODEMASK        DEFINE    0x1f (00001111)

NOINT           DEFINE    0xc0 (11000000)
  mrs r0,cpsr                       ;把状态寄存器的值送到 r0
  bic  r0,r0,#MODEMASK              ;把 cpsr 的低四位清零
  orr   r1,r0,#UNDEFMODE | NOINT
                    ;把 r0 的值置为:11011011 然后送给 r1
  msr       cpsr_cxsf,r1    ;把 r1 的值送到状态寄存器
       ;cpsr 现在的值是:
    ;cpsr[8:7]=11 相当于禁止 IRQ 和 FIQ 中断
    ;cprs[4:0]=11011 处理器的工作模式:未定义
  ldr    sp,=UndefStack        ;把堆栈的位置送给 SP
```

✧ARM 寄存器总结

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

- ✓ ARM 有 16 个 32 位的 registers(r0 到 r15).
- ✓ r15 充当程序计数器 PC, r14 (link register) 存储子程序的返回地址,r13 是堆栈地址.
- ✓ ARM 有一个当前程序状态 register:CPSR.
- ✓ 一些 registers(r13,r14)在异常发生时会产生新的 instances, 比如 IRQ 处理器模式,这时处理器使用 r13_irq 和 r14_irq
- ✓ ARM 的子程序调用是很快的,因为子程序的返回地址不需要存放在堆栈中.

➢ ARM Instructions

✧指令集概述

✓运算指令集

ADD	Add	[Rd]	Op1 + Op2
ADC	Add with carry	[Rd]	Op1 + Op2 + C
SUB	Subtract	[Rd]	Op1 - Op2
SBC	Subtract with carry	[Rd]	Op1 - Op2 + C – 1
RSB	Reverse subtract	[Rd]	Op2 - Op1
RSC	Reverse subtract with carry	[Rd]	Op2 - Op1 + C – 1
MUL	Multiply	[Rd]	Op1 x Op2
MLA	Multiply and accumulate	[Rd]	Rm x Rs + Rn

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

✓ 位操作指令集

AND	Logical AND	[Rd]	Op1	Op2
ORR	Logical OR	[Rd]	Op1	Op2
EOR	Exclusive OR	[Rd]	Op1	Op2
BIC	Logical AND NOT	[Rd]	Op1	NOT Op2

✓ 比较测试指令集

CMP Compare

;Set condition codes on Op1 - Op2

CMN Compare negated

;Set condition codes on Op1 + Op2

TS Test

;Set condition codes on Op1 Op2

TEQ Test equivalence

;Set condition codes on Op1 Op2

✓ 数据移动, 访问指令集

MOV	Move	[Rd]	Op2
MVN	Move negated	[Rd]	NOT Op2
LDR	Load register	[Rd]	[M(ea)]
STR	Store register	[M(ea)]	[Rd]
LDM	Load registers multiple		

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

;Load a block of registers from memory

STM Store registers multiple

;Store a block of registers in memory

ADDS r3,r1,r2 与 ADD r3,r1,r2 有何区别?

如果有S后缀,要根据计算结果修改状态寄存器的条件码标志位

✧指令的条件执行

几乎所有的ARM指令均可包含一个可选的条件码,只有在CPSR中的条件码标志满足指定的条件时,指令才执行.我们要作的就是把条件定义附加加在指令的后面.

如:

Add r1,r2,r3 这条指令无条件地执行如下操作:

[r1] ← [r2] + [r3]

然而 对于下面的指令就不是无条件执行了

ADDEQ r1,r2,r3

这条指令只有当 CPSR 里的 Z-bit 位为 1 时才执行加法操作

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

IF Z = 1 THEN [r1] ← [r2] + [r3]

举例

Example 1:

IF x = y THEN p = q + r ELSE IF x < y THEN p = q - r

假设: x, y, p, q, r 分别存在 r0, r1, r2, r3,r4 中

```
CMP r0,r1          ;compare x and y
ADDEQ r2,r3,r4     ;IF x = y THEN p = q + r
SUBLS r2,r3,r4     ;ELSE IF x < y THEN p = q – r
```

Example 2:

IF (a = b) AND (c = d) THEN e= e + 1;

假设: a, b, c, d, e 分别存在 r0, r1, r2, r3,r4 中

```
CMP r0,r1          Compare a and b
CMPEQ r2,r3        If a = b THEN compare c and d
ADDEQ r4,r4,#1     if c = d then increment e by 1
```

✧程序分支

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

- ✓ 简单的无条件转移unconditional branch

```
B Next    ;branch to "Next"
```

- ✓ 有条件转移

如果CPSR的 Z-bit是0时,跳转.如下例:

```
    MOV R0,#20     ;load the loop counter R0 with 20
Next               ;body of loop
    SUBS R0,R0,#1  ;decrement loop counter
    BNE Next       ;repeat until loop count = zero
```

- ✓ 带链接转移(类似子程序调用)

```
BL Next   ;branch to "Next" with link
```

这种情况下用R14来保存R15的值,然后在跳转到子程序处.

```
[r14] ← [PC]  ;copy program counter to link register
[r15] ← Next   ;jump to "Next"
```

✧ Data Movement Memory Reference Instructions

- ✓ 从一个Register Copy数据到另一个Register

```
MOV r0,#0              ;[r0] ← 0; Clear r0
MOV r0,r1, LSL #4      ;[r0]← [r1] * 16
MOVNE r3,r2, ASR #5    ;IF Z = 0 THEN [r3] ←[r2]/32
MOVS r0,r1, LSL #4     ;[r0] ← [r1] * 16; update
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
                                    condition codes
MVN r0,#0                   ;[r0]← -1; the 1's
                                    complement of 0 is 111...1
MVN r0,r0                   ;[r0] ← [r0]; complement the
                                     bits of r0
MVN r0,#0xF                 ;[r0] ← 0xFFFFFFF0
```

✓ 存储器与寄存器交换数据

```
LDR r0,[r1]       ;load r0 with the word pointed at by r1
STR r2,[r3]       ;store the word in r2 in the location
                    pointed at by r3
LDR r0,[r1,#8]    ;effective address = [r1] + 8, r1 is
                    unchanged
 LDR r0,[r1,#8]!
                  ;effective address = [r1] + 8, [r1]←[r1] + 8
 LDR r0,[r1],#8
                  ;effective address = [r1], [r1] ←[r1] + 8
```

➢ **Examples**

✧向量乘

$$s = A \cdot B = a1 \cdot b1 + a2 \cdot b2 + a3 \cdot b3 + ... + an \cdot bn$$

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        MOV r4,#0       ; clear initial sum in r4
        MOV r5,#24      ; load loop counter with n (assume 24 here)
        ADR r0,A        ; r0 points at vector A
        ADR r1,B        ; r1 points at vector B
Next
        LDR r2,[r0],#4  ; Repeat: get Ai and update pointer to A
        LDR r3,[r1],#4  ; get Bi and update pointer to B
        MLA r4,r2,r3,r4 ; s = s + Ai x Bi
        SUBS r5,r5,#1   ; decrement loop counter
        BNE Next        ; repeat n times
```

✧字符串比较

比较两个16-byte的串

```
        ADR r0, String1     ; r0 points to the first string
        ADR r1, String2     ; r1 points to the second string
        LDMIA r0, {r2-r5}   ; get first 16-byte string in r2 to r5
        LDMIA r1, {r6-r9}   ; get second 16-byte string in r6 to r9
        CMP r2, r6          ; compare two 4-byte chunks
        CMPEQ r3, r7        ; if previous 4 bytes same then
                                compare next 4
        CMPEQ r4, r8        ; and so on
        CMPEQ r5, r9
        BEQ Equal           ; if final 4 same then strings are equal
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

NotEq

…… .; if we end here then string not same

Equal

…….

✧子程序调用

SUB1. ; This is the first subroutine

……

STMFD r13, (r0-r4, lr) ; save working registers and link register

BL SUB2 ; call subroutine SUB2

……

……

LDMFD r13, (r0-r4, pc) ; restore working registers and return

.

…….

SUB2 ; a subroutine called from SUB1

……

MOV pc, lr ; return (copy link register to PC)

子程序调用通过带连接的分支指令 BL 实现,她把返回地址保存

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

在 R14(LR)里.如果一个子程序调用另一个子程序,我们必须在 R14 被第二个子程序调用覆盖前,把她保存在堆栈里.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

第 2 章

第二章:引导代码分析

➢前言

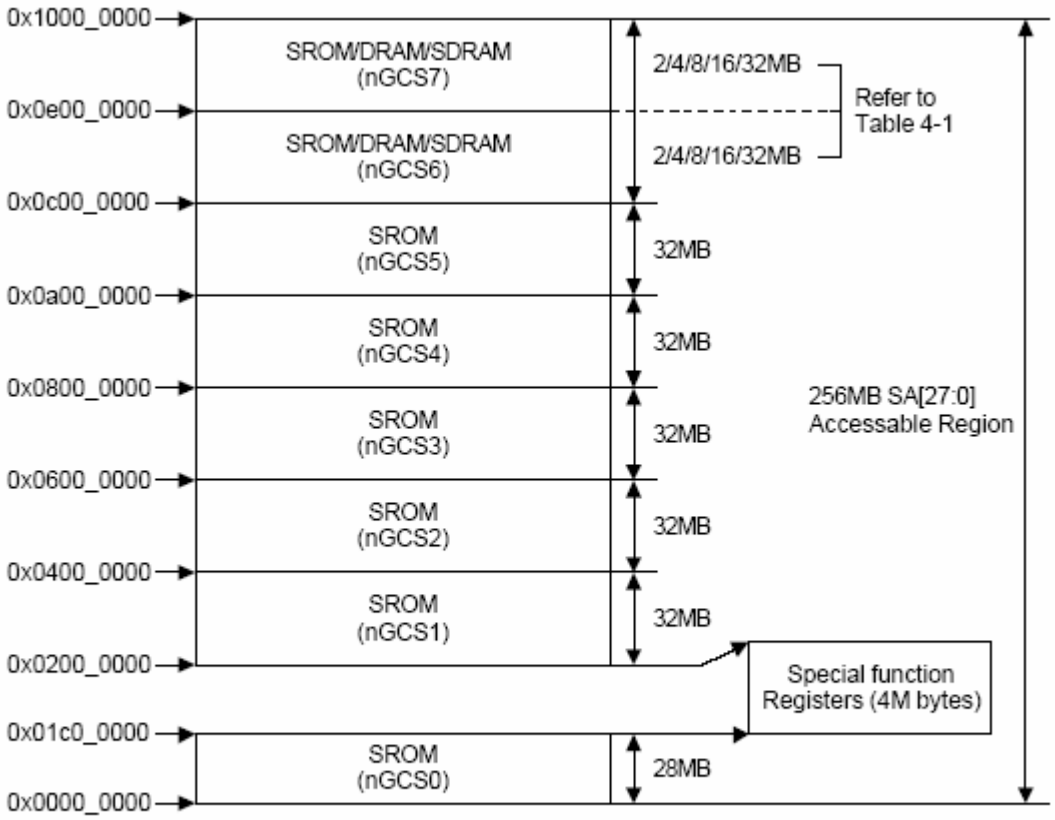
学习ARM，都不可避免地要了解系统引导，引导程序是系统加电后运行的第一段软件代码，本章将以S3C44B0X为目标来分析引导程序,希望能对朋友有所帮助。

➢概述

✧与 BOOT 相关硬件:FLASH ROM

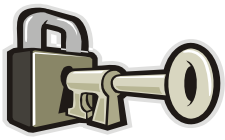
系统启动以后，S3C44B0X的编址方式如下图:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



NOTE: SROM means ROM or SRAM type memory

从上图可以看出，S3C44B0X 的最大寻址空间为 32M*8=256M. 和 S3C4510B 不同,S3C44B0X 处理器不支持 Memory Remap.



挂接BOOT ROM

S3C44B0X 自身不具有 ROM，因此必须外接 ROM 器件来存放掉电后仍需要保存的代码和数据,比如 Boot ROM

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

image.

在 S3C44B0X 芯片上有 8 个管脚 nGCS[7:0]用来做外接 ROM 芯片选择,如果 ROM 芯片的使能管脚 nCE 和 S3C44B0X 的管脚 nGCS0 相连(见下图), 就相当于把 ROM 映射在 S3C44B0X 的 bank0 地址空间(见下面的电路图).

从上面的地址映射图可以看出 BANK0 的起始地址为 0x0, S3C44B0X 在系统复位的时候, 处理器从 0x0 地址开始执行程序.所以,S3C44B0X 系统加电后运行的第一段软件代码就是映射在 BANK0 的 ROM 里存放的代码.

我们要做的就是:

- ✓ 把 FLASH ROM 如下图接在 BANK0
- ✓ 把 BOOT image 烧在 FLASH ROM

这样一个裸设备就可启动了.

Boot ROM 的数据总线宽度(Data Bus Width)由 S3C44B0X 的管脚 OM[1:0]的取值来决定:

Table 4-1. Data Bus Width for ROM Bank 0

OM[1:0]	Data Bus Width
00	8-bit (byte)
01	16-bit (half-word)
10	32-bit (word)
11	Test Mode

电路原理图:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

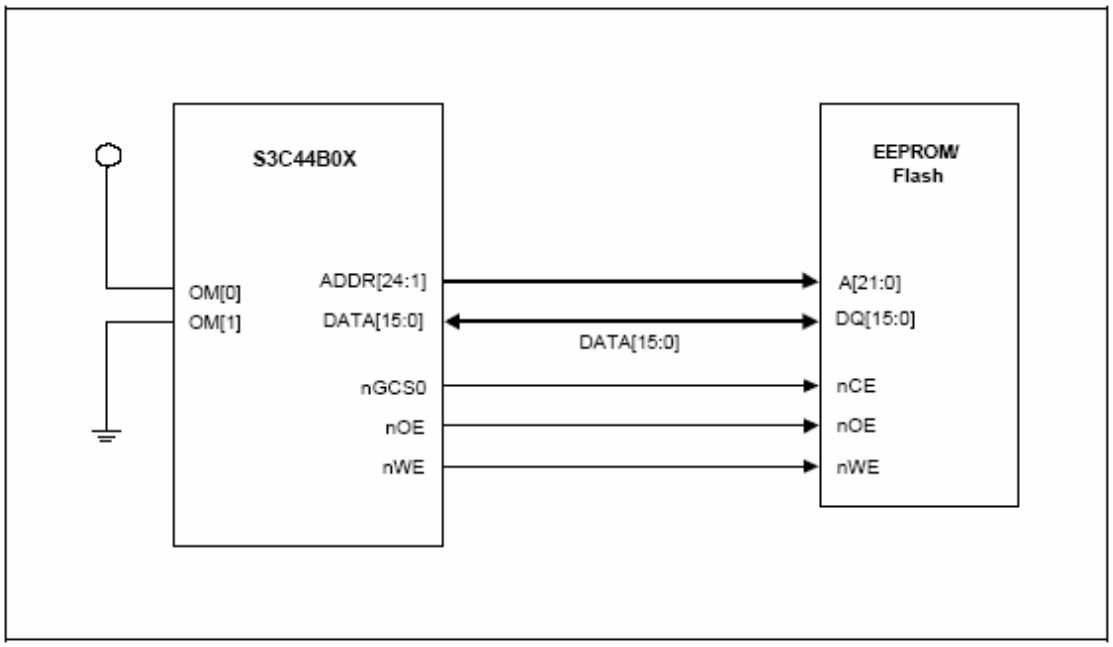


Figure 4-3. The Half-Word Boot ROM Design with Half-Word EEPROM/Flash

如何把 BOOT image 烧在 FLASH ROM?

✧BOOT 的主要功能

- **建立中断异常矢量表**

异常中断矢量表(Exception Vector Table)是Boot

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

与操作系统发生联系的地方。即使操作系统内核已经得到处理器的控制权运行，一旦发生中断，处理器还是会自动跳转到从0x0地址开始的异常中断矢量表中的某个位置（依据于中断类型）读取指令然后运行。S3C44B0X处理器不支持Memory Remap，这意味着，一旦发生中断，程序必须到Flash Memory中的Exception Vector Table里去走一圈.

对于S3C44B0X,地址0x0处的异常中断矢量表只简单地包含跳转指令,具体的中断处理由操作系统的中断处理程序来完成.

如果在Boot执行的全过程中都不必响应中断，那么如下的中断矢量表就可以OK.

```
b ResetHandler      ;for debug
b HandlerUndef      ;handlerUndef
b HandlerSWI        ;SWI interrupt handler
b HandlerPabort     ;handlerPAbort
b HandlerDabort     ;handlerDAbort
b .                 ;handlerReserved
b HandlerIRQ
b HandlerFIQ
```

Pabort & Dabort

对于 ARM 处理器来说，由于其内部使用了哈佛结构---独立的数据的指令总线,因此，在数据/指令的读取过程中产生的异常也

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

就很自然地可以区分开来，本质上而言，这些异常都是同属于存储访问失败产生的异常，因此这些异常都由 MMU 相关，在 ARM 手册中 DataAbort 和 PrefetchAbort 都称为 Memory abort

如果Boot功能要求使用中断，BOOT如何修改？

◆ 初始化堆栈

ARM7TDMI支持6种Operation Mode：

- ✓ User Mode
- ✓ FIQ Mode
- ✓ IRQ Mode
- ✓ Supervisor Mode
- ✓ Abort Mode
- ✓ Undefined Mode

BOOT需要为每种模式建立堆栈,这需要初始化其程序状态寄存器(SPSR)和堆栈指针.

系统需要初始化那些堆栈取决于用户使用了那些中断,以及系统需要处理那些错误类型.一般来说管理者堆栈必须设置,如果使用了IRQ中断,则IRQ堆栈也必须初始化.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

◆ 初始化硬件

软件运行离不开硬件,就像男人离不开女人一样(呵呵,和尚除外),BOOT必须要对硬件进行初始化.硬件的初始化通过配置特殊控制寄存器来完成,包括下面几个部分:

- ✓ 关Watchdog Timer
- ✓ 屏蔽所有的中断

为中断提供服务通常是 OS 设备驱动程序的责任. 因此Boot的执行全过程中可以不必响应任何中断。中断屏蔽可以通过写 CPU 的中断屏蔽寄存器或状态寄存器(ARM 的 CPSR 寄存器）来完成

- ✓ 初始化PLL和时钟

PLL的输出频率要就是处理器的工作主频.

- ✓ 初始化RAM

S3C44B0X使用一组专用的特殊功能寄存器来控制外部存储器的读/写操作,通过对该组特殊功能寄存器编程,可以设定外部数据总线宽度,访问周期,定时的控制信号（例如RAS和CAS）等参数.这些主要要通过设置13 个从0x1C80000(BWSCON)开始的寄存器(MEMORY CONTROLLER SPECIAL REGISGERS)来完成.

- ✓ 复制RW到DRAM，将ZI段清零

一个ARM由RO，RW和ZI三个段组成，其中RO为代码段，RW是已初始化的全局变量，ZI是未初始化的全局变量(与TEXT，DATA和BSS相对应)。

RO段是只读的，在运行的时候不可以改变，所以，

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

在运行的时候，RO段可以驻留在Flash里。

RW段是可以读写的，所以，在运行的时候必须被装载到SDRAM或者SRAM里,所以Boot要将RW段复制到RAM中，并将ZI段清零，以保证程序可以正确运行。

编译器使用下列符号来记录各段的起始和结束地址:

|Image$$RO$$Base| : RO段起始地址

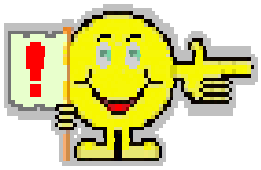
|Image$$RO$$Limit| : RO段结束地址加1

|Image$$RW$$Base| : RW段起始地址

|Image$$RW$$Limit| : ZI段结束地址加1

|Image$$ZI$$Base| : ZI段起始地址

|Image$$ZI$$Limit| : ZI段结束地址加1



这些标号的值是根据链接器中对ro-base和rw-base的设置来计算得到的.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

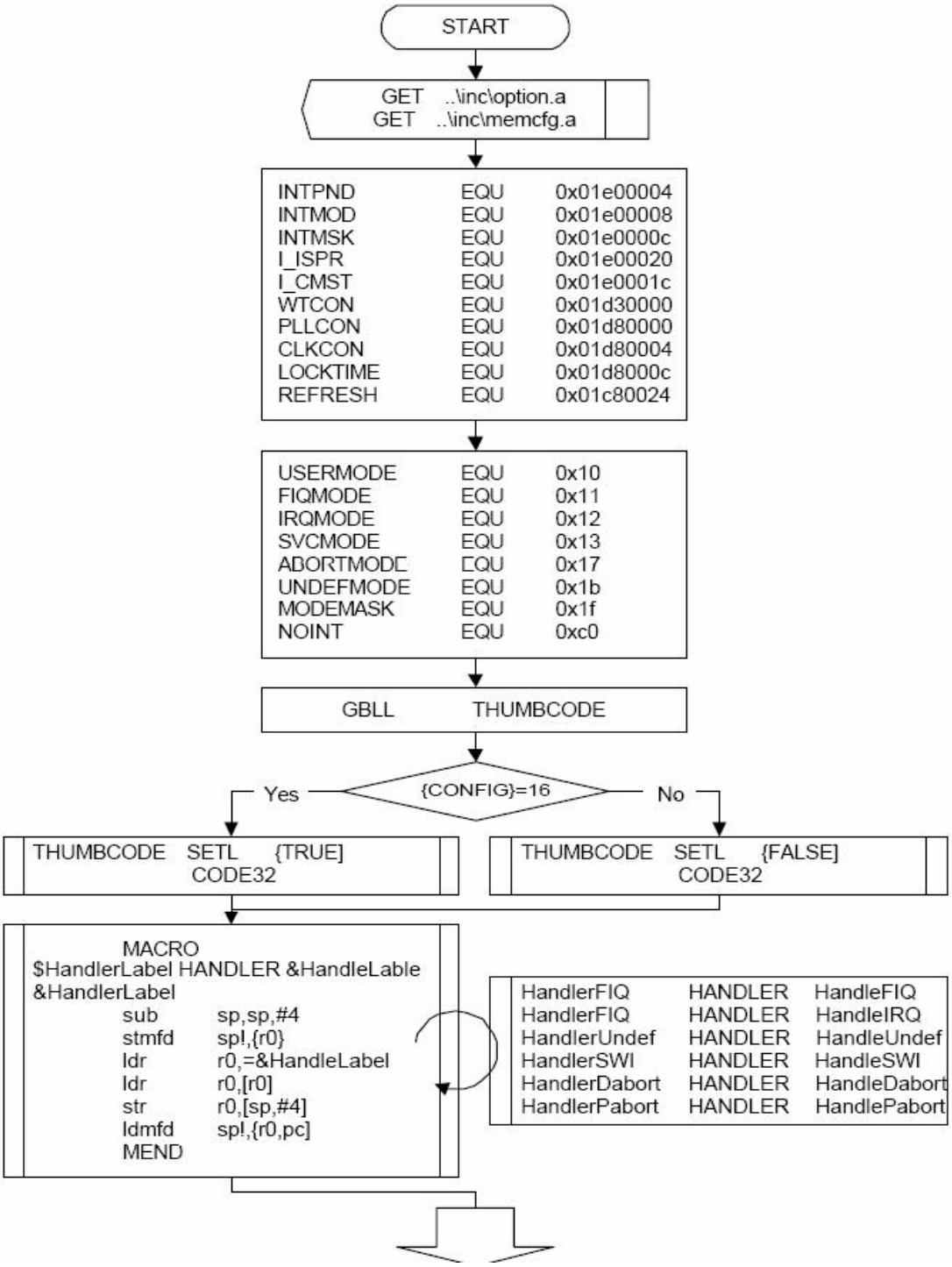
◆跳转到 C 语言程序，开始第二阶段的初始化和系统引导

➢执行流程及代码分析

◆ 参数初始化

程序的最开始的地方做变量定义等初始化工作,BOOT 也不例外.流程图如下:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

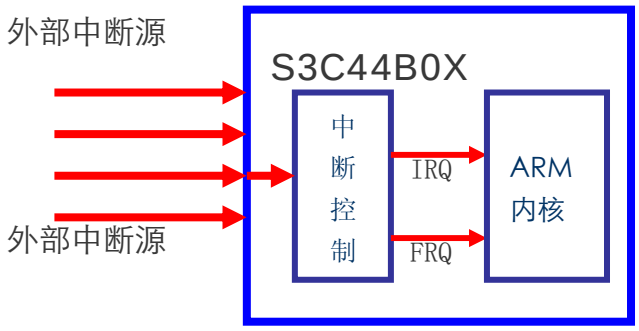


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

◆ 中断处理

✓ 中断源

外界有很多很多的中断源，S3C44B0X用一个中断控制器来管理各种原因产生的中断。 如下图:



当外部中断源产生中断时,他会触发与中断控制器的那根信号线，中断控制器收到这个信号后会检查一下这个中断是否被允许和是否被屏蔽,如果没有的话,就给她排一个处理的优先级,当轮到这个中断时,触发ARM的中断信号通知ARM内核.然后ARM内核就会去ＦＬＡＳＨ访问中断向量表.

ARM要求中断向量表必须放置在从０ｘ0地址开始、连续8×4字节的空间内，每当一个中断发生后，处理器会自动跳转到从0x0地址开始的异常中断矢量表中的某个位置（由中断类型来确定）读取指令然后运行.

✓ 中断处理模式

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

S3C44B0X 的中断控制器支持两个中断处理模式:

普通中断模式(NON-VECTORED INTERRUPT MODE)

向量中断模式(vectored interrupt mode)

我们可以通过配置中断控制寄存器

INTCON(0x01E00000)选择使用那一个模式:

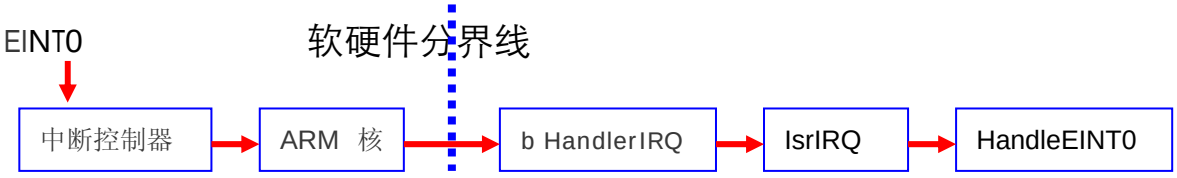
INTCON[2]: This bit disables/enables vector mode for IRQ
1: = Non-vectored interrupt mode
0: = Vectored interrupt mode

下面代码可以配置为向量中断模式

INTCON=0x1;

不同的中断模式会有不同的处理方式. 如图所示:

- 普通中断模式:



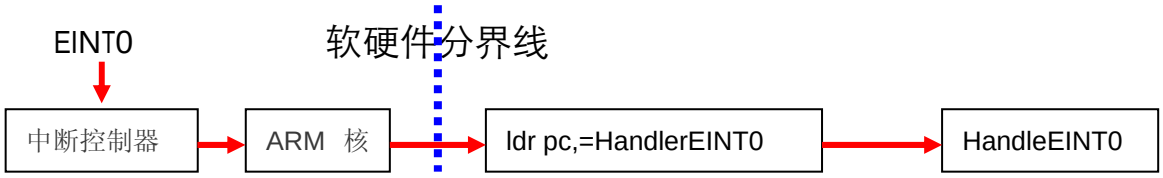
CPU从向量表中的地址值0x18(IRQ)处执行,该处指令是

b HandlerIRQ,

然后通过IsrIRQ例程计算出相应HandleEINT0的值，这个值为中断处理程序的起始地址，再将此值加载到PC。

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

- 向量中断模式



和普通中断模式比较，向量中断模式下少了一项工作：不执行IsrIRQ子程序。他直接找到了中断服务程序的起始地址,由此可见向量中断模式下处理中断要快一点.

向量中断模式

在向量中断模式下，当中断发生时，CPU会跳转到向量表中相应中断类型的表项,直接把中断服务例程的起始地址送到PC

例如：若产生EINT0中断，那么CPU会自动跳转到0x20地址处（而不是0x18），该处的指令是:

ldr pc,=HandlerEINT0,

这条指令把中断服务例程的起始地址送到PC。我们要做的就是把相应的中断服务程序放到这个地址空间,当中断发生时,执行之.

```
HandlerEINT0
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
sub sp,sp,#4
    ;decrement sp(to store jump address)
stmfd sp!,{r0}
     ;PUSH the work register to stack(lr does't push because it
      return to original address)
ldr r0,=HandleEINT0
    ;load the address of HandleEINT0 to r0
ldr r0,[r0]
       ;load the contents(service routine start address) of
  HandleEINT0
str r0,[sp,#4]
    ;store the contents(ISR) of HandleEINT0 to stack
ldmfd sp!,{r0,pc}
     ;POP the work register and pc(jump to ISR)
```

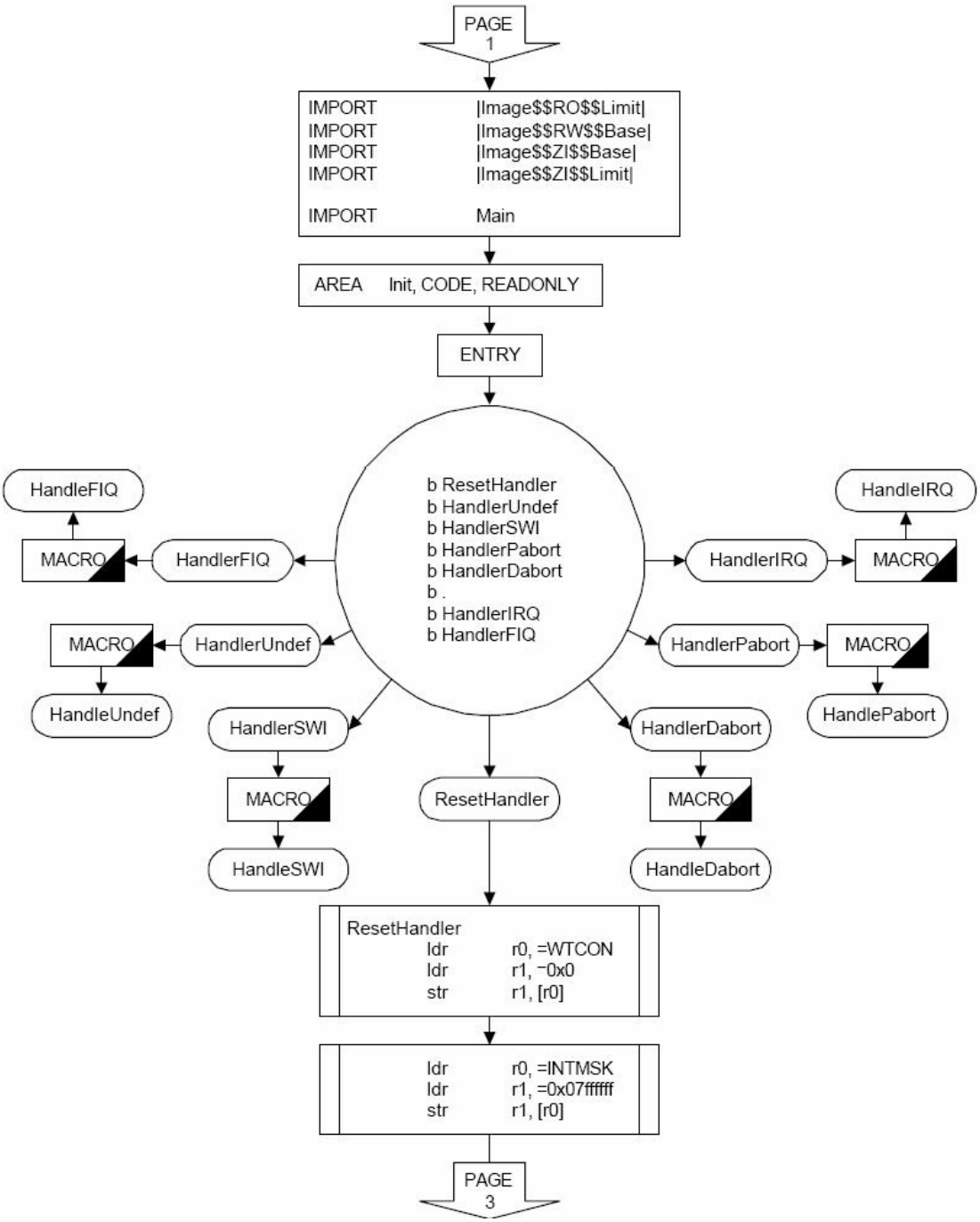
普通中断模式

如果为普通中断模式,CPU跳转到向量表中的表项0x18(IRQ)处开始执行.该处指令为

b HandlerIRQ ,

流程图如下:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



HandlerIRQ

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
sub sp,sp,#4
    ;decrement sp(to store jump address)
stmfd sp!,{r0}
    ;PUSH the work register to stack(lr does't push
    because it return to original address)
ldr r0,=HandleIRQ
    ;load the address of HandleEINT0 to r0
ldr r0,[r0]
    ;load the contents(service routine start
  address) of HandleEINT0
str r0,[sp,#4]
    ;store the contents(ISR) of HandleEINT0 to stack
ldmfd sp!,{r0,pc}
    ;POP the work register and pc(jump to ISR)
```

显然，这时候程序跳到了HandleIRQ处运行 ，那么HandleIRQ的值现在是多少呢?也就是说程序跳向何方?

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

看看源程序中这一段代码:

```
;*************************************************
;* Setup IRQ handler       *
;*************************************************
ldr       r0,=HandleIRQ   ;This routine is needed
ldr       r1,=IsrIRQ          ;if there isn't 'subs pc,lr,#4' at
                                0x18, 0x1c
str       r1,[r0]
```

从上面这段代码我们可以看到：在boot初始化的时候，标号为HandleIRQ的变量被赋了值，被赋予的值就是IsrIRQ.

由此可以分析出,当非向量中断发生时, CPU跳转到IsrIRQ处执行IsrIRQ后面的代码,执行的结果就得到了当前中断服务程序的起始地址.

和向量中断模式比较，非向量中断多运行了一段IsrIRQ代码来判断中断源,计算中断服务程序的起始地址。

BOOT为啥要这样作啊?

因为所有的非向量中断发生时,都跳到了0x18处:

```
b HandlerIRQ
```

BOOT不知道究竟发生了那个中断,所以为了找到准确的中断服务程序,必须要用点功夫计算一下.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

I_ISPR register记录了当前的中断,所以BOOT在IsrIRQ代码中借用I_ISPR register和BOOT程序中的数据段定义成功地获得中断服务程序的地址.

IsrIRQ通过分析I_ISPR register 来判断中断源,然后根据中断源算出中断服务程序的起始地址.

注:I_ISPR 记录了当前的中断源

```
IsrIRQ      ; using I_ISPR register.
    Sub sp, sp, #4;        reserved for PC
    Stmfd sp!, {r8-r9}
    Ldr r9,=I_ISPR
    Ldr r9,[r9]
    Mov r8, #0x00
0
    Movs r9,r9,lsr #1
    Bcs %F1
    Add r8, r8, #4
    B %B0
1
    Ldr r9, =HandleADC
    Add r9, r9, r8
    Ldr r9, [r9]
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

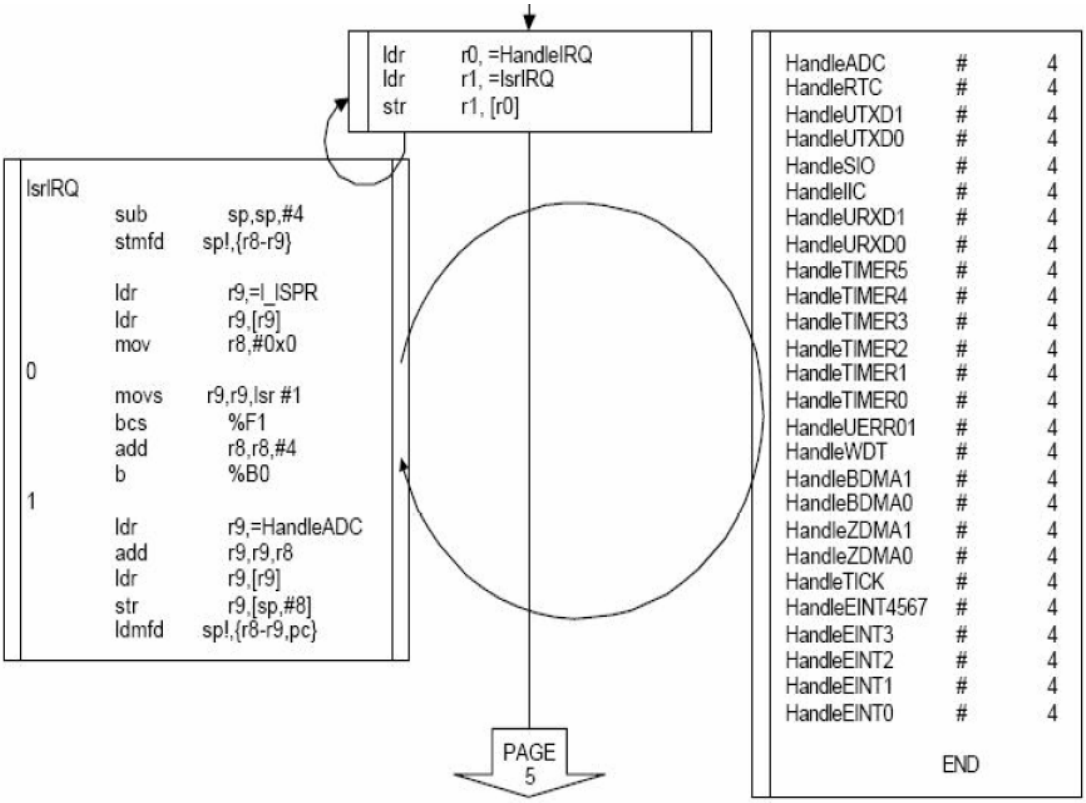
STR r9, [sp, #8]

Ldmfd sp!, {r8-r9, pc}

非向量中断通过执行IsrIRQ判断中断源，并计算出相应中断服务程序的起始地址HandleEINT0，然后将此地址加载到PC。

我们要做的就是在BOOT完了之后在实现OS的时候，把相应的中断服务程序放到HandleEINT0指定的这个地址空间,当INT0中断发生时,执行之.

流程图如下



编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

从BOOT源程序中数据段定义可以看出所有中断服务程序的起始地址以变量_ISR_STARTADDRESS的值为起点,如下图:

_ISR_STARTADDRESS	
_ISR_STARTADDRESS + 4	HandleReset
_ISR_STARTADDRESS + 8	HandleUndef
.....	.....
_ISR_STARTADDRESS + 33*4	HandleEINT0

那_ISR_STARTADDRESS的值是多少呢?

在BOOT开始部分包含一个option.a文件

```
GET ..\inc\option.a
```

从option.a文件我们可以找到_ISR_STARTADDRESS的定义:

```
_ISR_STARTADDRESS      EQU      0xc7fff00
```

注:0xc7fff00这个值可以根据你的需要修改

根据数据段定义的偏移量计算得到HandleEINT0的地址为

0xc7fff00+(33*4)十进制=0xc7fff84

注: (33*4)十进制=84(十六进制)

明确了中断服务程序的起始地址,我们可以在操作系统里用C语言实现为中断INT0服务的中断服务程序,然后把这个中断服务程序安装在_ISR_STARTADDRESS+0x84处.

✓ 定义中断服务程序的地址

```
#define pISR_EINT0 (*(unsigned *)(_ISR_STARTADDRESS+0x84))
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

- ✓ 实现中断服务程序

```
Void    Eint0(void)
{
 Bababa;
 }
```

- ✓ 把中断服务程序安装在 _ISR_STARTADDRESS+0x84 处

```
pISR_EINT0=(int)Eint0;
```

详细的描述见第三章[中断服务程序编写]

```
BOOT 数据段定义:

AREA    RamData, DATA, READWRITE

^ (_ISR_STARTADDRESS-0x500)

UserStack       # 256     ;c1(c7)ffa00
SVCStack        # 256     ;c1(c7)ffb00
UndefStack      # 256     ;c1(c7)ffc00
AbortStack      # 256     ;c1(c7)ffd00
IRQStack        # 256     ;c1(c7)ffe00
FIQStack        # 0       ;c1(c7)fff00

^ _ISR_STARTADDRESS
HandleReset             # 4
HandleUndef             # 4
HandleSWI               # 4
HandlePabort            # 4
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
HandleDabort        # 4
HandleReserved      # 4
HandleIRQ           # 4
HandleFIQ           # 4

;Don't use the label 'IntVectorTable',
;because armasm.exe cann't recognize this label correctly.
;the value is different with an address you think it may be.
;IntVectorTable
HandleADC       # 4
HandleRTC       # 4
HandleUTXD1     # 4
HandleUTXD0     # 4
HandleSIO       # 4
HandleIIC       # 4
HandleURXD1     # 4
HandleURXD0     # 4
HandleTIMER5    # 4
HandleTIMER4    # 4
HandleTIMER3    # 4
HandleTIMER2    # 4
HandleTIMER1    # 4
HandleTIMER0    # 4
HandleUERR01    # 4
HandleWDT       # 4
HandleBDMA1     # 4
HandleBDMA0     # 4
HandleZDMA1     # 4
HandleZDMA0     # 4
HandleTICK      # 4
HandleEINT4567  # 4
HandleEINT3     # 4
HandleEINT2     # 4
HandleEINT1     # 4
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
HandleEINT0      # 4
END
```

◆ 初始化硬件

✓ 涉及的硬件

- 晶振电路

晶振电路用于向CPU及其他外围电路提供工作时钟。

- 锁相环(PHASE-LOCKED-LOOP)PLL

锁相环是一个相位误差控制系统。它比较输入信号和振荡器输出信号之间的相位差，从而产生误差控制信号来调整振荡器的频率，以达到与输入信号同频同相。

锁相环一般都提供了倍频、分频、相移三个功能，片内的PLL电路兼有频率放大和信号提纯的功能，因此，系统可以以较低的外部时钟信号获得较高的工作频率，以降低因高速开关时钟所造成的高频噪声.

- 看门狗计时器

看门狗定时器是系统软件崩溃后进行恢复的一种方法,一次WDT 超时溢出将产生一次器件复位.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

✓ 初始化分析

- 关闭看门狗计时器 watch dog timer

```
Register: WTCON
Address: 0x01D30000
ldr     r0,=WTCON       ;watch dog disable
ldr     r1,=0x0
str     r1,[r0]
```

- 屏蔽所有中断

为中断提供服务通常是 OS 设备驱动程序的责任. 因此在 Boot Loader 的执行全过程可以不必响应任何中断.

```
Register: INTMSK
Address: 0x01E0000C
ldr     r0,=INTMSK
ldr     r1,=0x07ffffff  ;all interrupt disable
str     r1,[r0]
```

- 配置时钟

相关寄存器:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

Register :PLLCON(PLL configuration Register)

Address: 0x01D80000

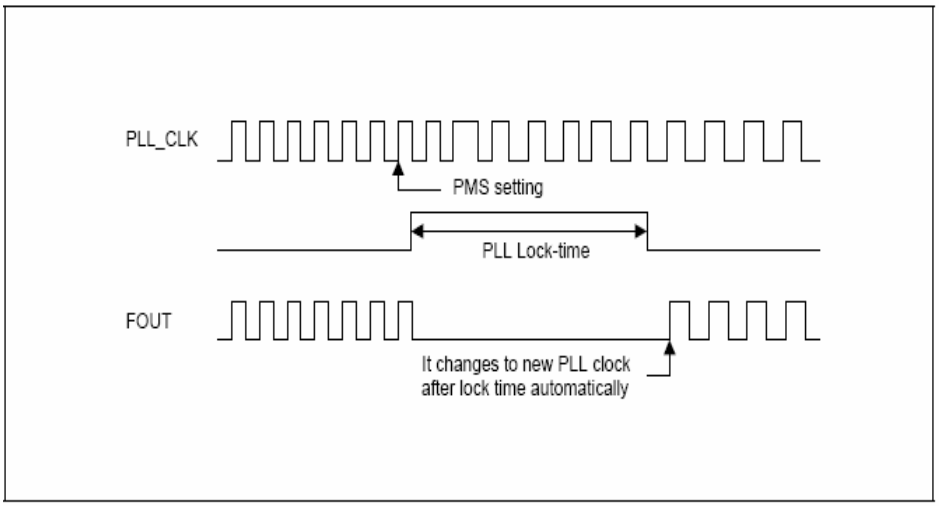
Register: CLKCON(Clock generator control)

Address: 0x01D80004

Register: LOCKTIME (PLL lock time count)

Address: 0x01D8000C

✧ PLL 稳定过渡时间配置



✧ 设置主频

PLLCON Register 用来配置主频，可以通过修改这个 Register 的内容达到修改配置主频的目的，下面这个

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

代码把主频配置为 60MHZ

```
;M_DIV  EQU   0x34
;P_DIV  EQU   0x3
;S_DIV  EQU   0x1
[ PLLONSTART
  ldr r0,=PLLCON     ;temporary setting of PLL
  ldr r1,=((M_DIV<<12)+(P_DIV<<4)+S_DIV)
          ;Fin=10MHz,Fout=60MHz
  str r1,[r0]
]
```

主频计算公式:

PLLCON[19:12]= M_DIV(Main divider control)

PLLCON[9:4]= P_DIV(Pre-divider control)

PLLCON[1:0]= S_DIV(Post divider control)

*Fpllo = (m * Fin) / (p * 2**s)*

m = (MDIV + 8), p = (PDIV + 2), s = SDIV

代入上面的值:

m=0x34+8=0x3c,p=0x3+2=5,s=1

*Fpllo=(0x3c*10)/(5*2)=0x3c=60MHZ*

✧ **给外设提供时钟**

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

对于电池驱动的SoC芯片，已不能再只考虑它优化空间的两个方面——速度（performance）和面积（cost），而必须要注意它已经表现出来的且变得越来越重要的第三个方面——功耗,这样才能延长电池的寿命和电子产品的运行时间

S3C44B0X的电源管理模块包含五个模式:

- ✓ Normal mode
- ✓ Slow mode
- ✓ Idle mode
- ✓ Stop mode
- ✓ SL Idle mode for LCD,

使用这些模式的目的是根据芯片不同的应用,动态地为CPU和外设提供时钟源,从而达到控制电源损耗的目的.

正常模式(The Normal mode)下,同时向CPU和外设提供时钟,这种模式下电源损耗最大.

把CLKCON设为0x7ff8,相当于设为正常模式Normal mode.

```
ldr     r0,=CLKCON
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
ldr     r1,=0x7ff8    ;All unit block CLK enable
str     r1,[r0]
```

● 初始化 RAM

CPU使用一组专用的特殊功能寄存器来控制外部存储器的读/写操作，通过对该组特殊功能寄存器编程，可以设定外部数据总线宽度,访问周期,定时的控制信号（例如RAS和CAS）等参数.

这主要通过设置 13 个从 1c80000(BWSCON)开始的特殊寄存器来设置.

0x1c80000 为特殊寄存器的首地址.

- ✓ BWSCON(0x1c80000): 配置总线宽度 data Bus Width :8-bit,16-bit, 32-bit
- ✓ BANKCON0-7(0x1c80004-0x1c80020):
  - 访问周期[10:8] Access cycle
  - 存储器的类型[16:15](Bank6,Bank7)
  - 定时Trcd的控制信号

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

Trcd: RAS to CAS delay

这是用来设定从RAS信号到CAS信号所需的时间周期(Clock)。当 CPU 要从内存读取或写入数据时，必须送出一个正确的地址信号，而地址是由行、列交错而成，所以又分为 RAS(列地址信号)和 CAS(行地址信号)，由于是先寻址 RAS、再寻址 CAS，因此中间就有延迟时间，建议先设成 2 个 Clock(2T)，让 SDRAM 能快点将地址寻址完毕，这样可以将内存性能提高；

Tcas: CAS pulse width

Tcp: CAS pre-charge

CAS 表示列地址寻址（Column Address Strobe or Column Address Select），

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

RAS 表示行地址寻址（Row Address Strobe ）

✓ REFRESH(0x01C80024):DRAM/SDRAM刷新配置:

Trp[21:20]: DRAM/SDRAM RAS pre-charge Time

设定当 RAS(列地址信号)需要重新寻址时，要隔多久时间才能开始下次的寻址动作，通常被视为 RAS 的充电时间，理论上是越短越好。其选项值与上面讲到的一样，所以建议将其设定为 2T(2 个 Clock 周期)，这样可以将内存性能提高；若发现系统运行不稳定，再将其改回 3T。

Trc [19:18] SDRAM RC minimum Time

这个参数用来控制内存的行周期时间。tRC 决定了完成一个完整的循环所需的最小周期数，也就是从行激活到行充

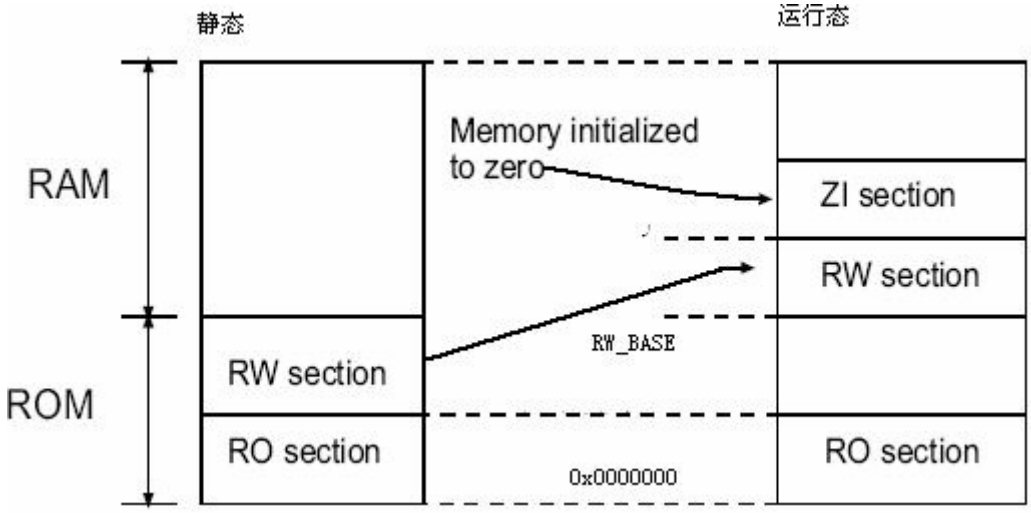
编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

电的时间

```
;*****************************************************
    ;* Set memory control registers     *
;*****************************************************
    ldr       r0,=SMRDATA
    ldmia     r0,{r1-r13}
    ldr       r0,=0x01c80000    ;BWSCON Address
    stmia     r0,{r1-r13}
```

● 复制 RW 到 DRAM，将 Zi 段清零

为了保证程序可以正确运行,在运行的时候必须把可读写的 RW 段被装载到 SDRAM 或者里.复制前后代码和数据段的分部如下图所示:



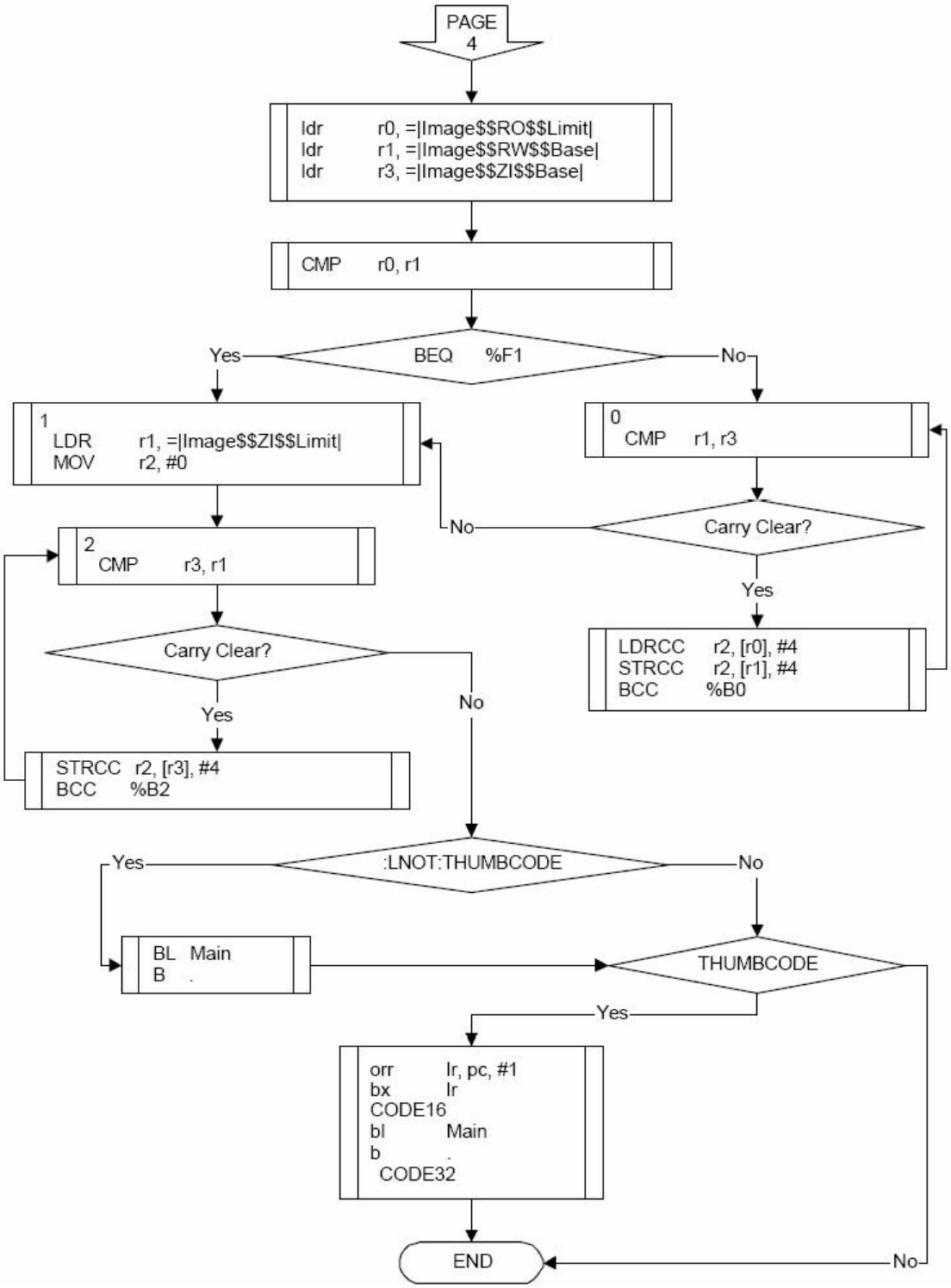
****

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;*   Copy and paste RW data/zero initialized data        *
;*****************************************************
LDR       r0, =|Image$$RO$$Limit| ; Get pointer to ROM data
LDR       r1, =|Image$$RW$$Base| ; and RAM copy
LDR       r3, =|Image$$ZI$$Base|
    ;Zero init base => top of initialised data
CMP       r0, r1          ; Check that they are different
BEQ       %F1
0
CMP       r1, r3          ; Copy init data
LDRCC     r2, [r0], #4    ;--> LDRCC r2, [r0] + ADD r0, r0, #4
STRCC     r2, [r1], #4    ;--> STRCC r2, [r1] + ADD r1, r1, #4
BCC       %B0
1
LDR       r1, =|Image$$ZI$$Limit| ; Top of zero init segment
MOV       r2, #0
2
CMP       r3, r1       ; Zero init
STRCC     r2, [r3], #4
BCC       %B2
```

流程图如下:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

◆ 跳转到 C 语言程序，开始第二阶段的初始化和系统引导.

源程序:

```
  [ :LNOT:THUMBCODE
    BL  Main          ;Don't use main() because ......
    B   .
  ]

[ THUMBCODE           ;for start-up code for Thumb mode
    orr     lr,pc,#1
    bx      lr
  CODE16
    bl      Main      ;Don't use main() because ......
    b       .
  CODE32
]
```

◆ 初始化堆栈

ARM7TDMI支持6种Operation Mode:

- ✓ User Mode
- ✓ FIQ Mode

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

- ✓ IRQ Mode
- ✓ Supervisor Mode
- ✓ Abort Mode
- ✓ Undefined Mode

需要为每个模式建立堆栈.系统堆栈初始化取决于用户使用了那些中断,以及系统需要处理那些错误类型.一般来说管理者堆栈必须设置.

对程序中需要用到的每一种模式都要给SP寄存器定义一个堆栈地址

如何定义堆栈地址?

方法是改变状态寄存器（CPSR）内的状态位，使处理器切换到不同的状态，然后给SP赋值。下面为具体的代码分析:

```
UNDEFMODE       DEFINE    0x1b (00011011)
MODEMASK        DEFINE    0x1f (00001111)
NOINT           DEFINE    0xc0 (11000000)
  mrs r0,cpsr                  ;把状态寄存器的值送到 r0
  bic r0,r0,#MODEMASK   ;把 cpsr 的低四位清零
  orr   r1,r0,#UNDEFMODE | NOINT
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;把 r0 的值置为:11011011 然后送给 r1

msr     cpsr_cxsf,r1        ;把 r1 的值送到状态寄存器

;cpsr 现在的值是:
;cpsr[8:7]=11 相当于禁止 IRQ 和 FIQ 中断
;cprs[4:0]=11011 处理器的工作模式:未定义
ldr   sp,=UndefStack      ;把堆栈的位置送给 SP
```

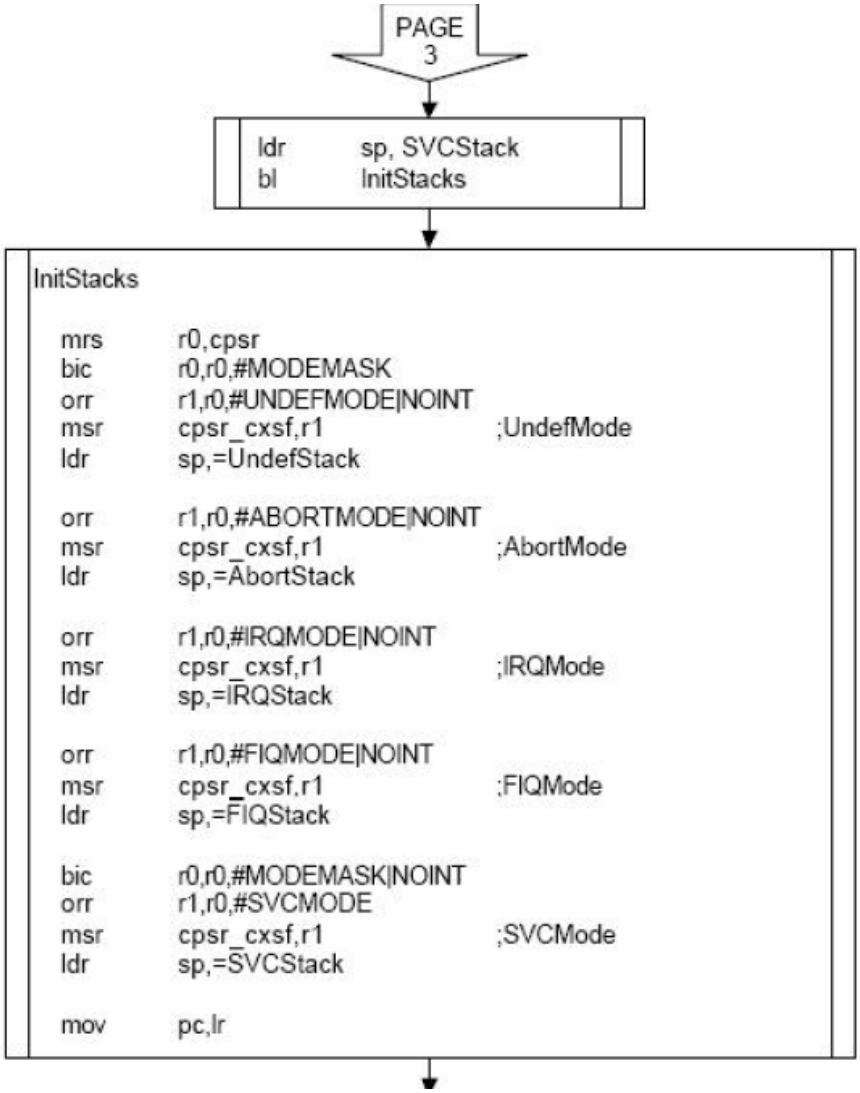
```
;*****************************************************
;* The function for initializing stack               *
;*****************************************************
InitStacks
  ;Don't use DRAM,such as stmfd,ldmfd.....
  ;SVCstack is initialized before
 ;Under toolkit ver 2.50, 'msr cpsr,r1' can be used instead of 'msr
  cpsr_cxsf,r1'

    mrs     r0,cpsr
    bic     r0,r0,#MODEMASK
    orr     r1,r0,#UNDEFMODE|NOINT
    msr     cpsr_cxsf,r1        ;UndefMode
    ldr     sp,=UndefStack

    orr     r1,r0,#ABORTMODE|NOINT
    msr     cpsr_cxsf,r1        ;AbortMode
    ldr     sp,=AbortStack
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        orr     r1,r0,#IRQMODE | NOINT
        msr     cpsr_cxsf,r1            ;IRQMode
        ldr     sp,=IRQStack

        orr     r1,r0,#FIQMODE | NOINT
        msr     cpsr_cxsf,r1            ;FIQMode
        ldr     sp,=FIQStack

        bic     r0,r0,#MODEMASK | NOINT
        orr     r1,r0,#SVCMODE
        msr     cpsr_cxsf,r1            ;SVCMode
        ldr     sp,=SVCStack

;USER mode is not initialized.
        Mov     pc,lr       ;The LR register may be not valid for the
                             mode changes.
```

流程图:

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



写到这里,BOOT 程序的分析算是告一段落.我们基本了解了 BOOT 的基本流程和相关知识,为下一步的初始化和跳转到操作系统打下了一个坚实的基础.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

第 3 章

第三章:中断服务程序编写

BOOT 执行结束后,通过下面的指令跳转到 C 程序:

BL Main

所以我们必须在跳到 Main 之后,实现中断服务程序,不然的话,中断发生后,系统就会像我们的国足队员一样要抽筋了.

本章就以外部中断 HandlerEINT4567 为例来介绍编写中断服务程序的流程和一些基本概念.

➢ 必需的变量定义

从第二章的中断部分我们得到中断服务程序的起始地址为:

_ISR_STARTADDRESS

所以我们在h头文件44b.h来用这个首地址加每个中断的

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

偏移量来定义中断服务程序的起始地址。

我们配置Port G为8个外部中断源输入,所以对Port G的控制寄存器也要定义。中断控制寄存器为必需品,当然不能少.

所以为了完成中断服务程序,我们必须在头文件里定义下面几部分内容

✧ **中断服务程序地址**

```
#define pISR_EINT4567   (*(unsigned *)(_ISR_STARTADDRESS+0x74))
```

✧ **I/O 端口**

```
#define rPCONG      (*(volatile unsigned *)0x1d20040)
#define rPDATG      (*(volatile unsigned *)0x1d20044)
#define rPUPG       (*(volatile unsigned *)0x1d20048)

#define rSPUCR      (*(volatile unsigned *)0x1d2004c)
#define rEXTINT     (*(volatile unsigned *)0x1d20050)
#define rEXTINTPND  (*(volatile unsigned *)0x1d20054)
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

✧ INTERRUPT 控制寄存器

```
#define rINTCON      (*(volatile unsigned *)0x1e00000)
#define rINTPND      (*(volatile unsigned *)0x1e00004)
#define rINTMOD      (*(volatile unsigned *)0x1e00008)
#define rINTMSK      (*(volatile unsigned *)0x1e0000c)

#define rI_ISPR      (*(volatile unsigned *)0x1e00020)
#define rI_ISPC      (*(volatile unsigned *)0x1e00024)
```

✧ EINT4567 的 Pending 位

```
#define BIT_EINT4567  (0x1<<21)
#define BIT_GLOBAL    (0x1<<26)
```

➢ 变量解释

为了能够更好的理解下面的内容,我们必须要对上面的定义做一下解释.

对于中断服务程序的地址,不用多讲废话.对 I/O 端口 INTERRUPT 控制寄存器有必要解释一下.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

✓ #define rPCONG (*(volatile unsigned *)0x1d20040)

阅读S3C44B0X的硬件说明文档我们发现,她有两组8位的输入/输出端口: Port D 和 PortG ,并且这些I/O口可以很容易地通过软件来配置,从而满足各种设计需求.

上面这个define就定义了PortG端口,我们打算配置她作为外部中断输入。

✓ #define rPDATG (*(volatile unsigned *)0x1d20044)

如果PortG 被配置成输出口(output ports) 数据写入这个寄存器;如果PortG 被配置成输入口(input ports) 从这个寄存器读数据

✓ #define rPUPG (*(volatile unsigned *)0x1d20048)

控制 PortG 口的上拉电阻是接通还是断开

✓ #define rSPUCR (*(volatile unsigned *)0x1d2004c)

控制数据总线的上拉电阻是接通还是断开

✓ #define rEXTINT (*(volatile unsigned *)0x1d20050)

这个用来配置中断是通过信号的上升沿触发,还是下降沿触发,是高电平触发, 还是底电平触发.

✓ #define rEXTINTPND (*(volatile unsigned *)0x1d20054)

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

EINT4, EINT5, EINT6, 和 EINT7这四个中断输入共享同一条中断请求线和中断控制器相连,那如何区分他们啊?不急,EXTINTPND寄存器就可以解决这个问题.

If EINT4 产生时, EXTINTPND[0] =1 (0001)

If EINT5 产生时, EXTINTPND[1] =1 (0010)

If EINT6 产生时, EXTINTPND[2] =1 (0100)

If EINT7 产生时, EXTINTPND[3] =1 (1000)

- ✓ #define rINTCON (*(volatile unsigned *)0x1e00000)

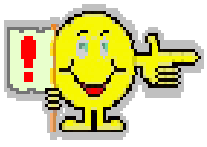
  中断控制寄存器,用来配置下面两个重要属性:

  使用向量还是非向量中断模式,

  使用普通中断(IRQ)还是快速中断(FRQ)

- ✓ #define rINTPND (*(volatile unsigned *)0x1e00004)

  这个寄存器的每一位对应一个中断源.例如当一个中断请求产生时,相应的位会置为1. 比如,当EINT4 产生时, INTPND[21]的值变为1



Interrupt Pending Register is a read-only register, so the service routine must clear the pending

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

condition by writing a 1 to I_ISPC or F_ISPC

- ✓ #define rINTMOD (*(volatile unsigned *)0x1e00008)

  为每个中断源设置中断模式,比如

  IF INTMOD[21]=1 EINT4 为 FIQ mode

  IF INTMOD[21]=0 EINT4 为 IRQ mode

- ✓ #define rINTMSK (*(volatile unsigned *)0x1e0000c)

  中断屏蔽寄存器,比他屏蔽后,中断就不能触发了.比如:

  If INTMSK[21]=1, EINT4被屏蔽

- ✓ #define rI_PSLV (*(volatile unsigned *)0x1e00010)
- ✓ #define rI_PMST (*(volatile unsigned *)0x1e00014)
- ✓ #define rI_CSLV (*(volatile unsigned *)0x1e00018)
- ✓ #define rI_CMST (*(volatile unsigned *)0x1e0001c)

  上面四个为优先级配置寄存器

- ✓ #define rI_ISPR (*(volatile unsigned *)0x1e00020)
- ✓ #define rI_ISPC (*(volatile unsigned *)0x1e00024)

  I_ISPR 确定当前正在服务的中断

  I_ISPC/F_ISPC 用来对INTPND/I_ISPR中的pending位清零,

  I_ISPC/F_ISPC 也通知中断控制器一个中断服务程序运行完成.在中断服务程序的最后必须把pending位清零.

手册上的英语描述的更清晰:

I_ISPR indicates the interrupt being currently serviced.

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

Although the several interrupt pending bits are all turned on,only one bit will be turned on.

I_ISPC/F_ISPC clears the interrupt pending bit (INTPND). I_ISPC/F_ISPC also informs the interrupt controller of the end of corresponding ISR (interrupt service routine). At the end of ISR(interrupt service routine), the corresponding pending bit must be cleared.

Interrupt Pending Register is a read-only register, so the service routine must clear the pending condition by writing a 1 to I_ISPC or F_ISPC

上面的解释说明是针对S3C44B0X的寄存器,对于其他的芯片一定会有区别,但他们都有共同的,相似的地方.

➢ 中断服务程序的实现

为了简单期间我们把中断服务程序放在在 main.c 文件里实现.

✧ 定义中断服务程序

```
void __irq Eint4567Isr(void);
volatile char which_int=0;//标识发生了那个中断,4,5,6 or 7?

void __irq Eint4567Isr(void)
 {
   which_int=rEXTINTPND;
       /*
       If EINT4  产生时, EXTINTPND[0] =1 (0001)1
       If EINT5  产生时, EXTINTPND[1] =1 (0010)2
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        If EINT6 产生时, EXTINTPND[2] =1 (0100)4
        If EINT7 产生时, EXTINTPND[3] =1 (1000)8
        所以上面语句执行的结果是:
        If EINT4 产生时 which_int=1
        If EINT4 产生时 which_int=2
        If EINT4 产生时 which_int=4
        If EINT4 产生时 which_int=8
       */

    rEXTINTPND=0xf;
      /* clear EXTINTPND reg
        The interrupt service routine must clear the interrupt
        pending condition(INTPND)  after clearing the
        external pending condition(EXTINTPND).
       */
     rI_ISPC=BIT_EINT4567;
      /* clear pending_bit */
}
```

✧主程序

```
int Main()
{
        unsigned int save_G,save_PG;
        rINTMOD=0x0;
        rINTCON=0x1;
          /*中断初始化
            rINTMOD=0x0 设置为 IRQ mode
            rINTCON=0x1 设置为向量中断模式
          */
       pISR_EINT4567=(int)Eint4567Isr;
       /*
       中断服务程序入口地址定位
        pISR_EINT4567
            0xc7fff74:
```

Eint4567Isr

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
把 Eint4567Isr 代码存放在地址为 0xc7fff74 开始的内存空间里
   */
  save_G=rPCONG;
  save_PG=rPUPG;
   /*保存原来 I/O 端口的配置 */
  rPCONG=0xffff;//EINT7~0
  rPUPG=0x0;     //pull up enable
  /*
    rPCONG=0xffff 把 PortG 配置为 8 个中断输入
    rPUPG=0x0       中断为上升沿触发
  */

  rINTMSK=~(BIT_GLOBAL|BIT_EINT4567);   ;
  /*开放屏蔽位*/
while(!which_int);
switch(which_int)
{
   case 1:
    Uart_Printf("EINT4 had been occured...\n");
    break;
   case 2:
    Uart_Printf("EINT5 had been occured...\n");
    break;
   case 4:
     Uart_Printf("EINT6 had been occured...\n");
     break;
   case 8:
     Uart_Printf("EINT7 had been occured...\n");
     break;
   default :
     break;
 }

rINTMSK=BIT_GLOBAL;
/* 屏蔽所有中断*/
rPCONG=save_G;
rPUPG=save_PG;
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
    which_int=0;
    Uart_Printf("\nrINTCON=0x%x\n",rINTCON);
}
```

✧ 中断服务子程序中关键的变量类型

- **Volatile**

Volatile在英文字典里“易变的，反复无常的（性格）”呵呵,为啥要定义一个反复无常的变量啊?

编译器有一种技术叫做数据流分析，分析程序中的变量在哪里赋值、在哪里使用、在哪里失效，分析结果可以用于常量合并，常量传播等优化。当他察觉到你的代码没有修改字段的值，她就有可能在你访问字段时提供上次访问的缓存值,这能够提高程序的效率.但有时这些优化会带来问题,不是我们程序所需要的,特别是对硬件寄存器操作的程序.这时我们可以用volatile关键字禁止做这些优化.

Volatile的作用：

- ✓ 不会在两个操作之间把volatile变量缓存在寄存器中。在多任务、中断处理程序中，变量可能被其他的程序改变，编

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

译器自己无法知道，volatile就是告诉编译器这种情况:变量已经变化,不要用缓存值哦。

- ✓ 不做常量合并、常量传播等优化，所以像下面的代码:

```
volatile int i = 1;
if (i > 0) ...
```

if的条件不会当作无条件真。因为一个变量定义为volatile类型就是说这变量可能会被意想不到地随时改变.

- ✓ 对volatile变量的读写不会被优化掉。如果你对一个变量赋值但后面没用到，编译器常常可以省略那个赋值操作，然而对有些硬件寄存器的处理是不能这样优化的

使用 volatile 变量的场合:

- ✓ 硬件寄存器通常也要加volatile说明，因为每次对它的读写都可能有不同意义.比如下面为中断屏蔽寄存器的定义

```
define rINTMSK  (*(volatile unsigned *)0x1e0000c)
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

✓ 在中断服务程序中修改的供其它程序检测的变量需要加volatile

如上面中断服务子程序Eint4567Isr中使用的which_int，定义为：

volatile char which_int=0;

如果在上面的变量定义中把volatile，改为static会产生啥结果？

```
Static char which_int=0;
void __irq Eint4567Isr(void)
{
        which_int=rEXTINTPND;
        rEXTINTPND=0xf;
        rI_ISPC=BIT_EINT4567;
}
   int Main()
{
.....................
.....................
      while(!which_int);
      switch(which_int)
      {
        case 1:
        Uart_Printf("EINT4 had been occured...\n");
          break;
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        case 2:
          Uart_Printf("EINT5 had been occured...\n");
          break;
        case 4:
          Uart_Printf("EINT6 had been occured...\n");
          break;
        case 8:
          Uart_Printf("EINT7 had been occured...\n");
          break;
        default :
            break;
         }
          ..........................
        }
```

程序的本意是希望 int4,int5,int6,int7 中断产生时，中断服务程序给 which_int 赋值,在 Main 中根据 which_int 的值调用 Uart_Printf 函数打印出相关信息. 但是，由于编译器判断在 Main 函数里面没有修改过 which_int，因此可能只执行一次从 which_int 到寄存器的读操作，然后每次 switch 判断都只使用这个寄存器里面的“which_int 副本”，导致 Uart_Printf 永远也不会被调用。如果将将变量用 volatile 修饰，则编

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

译器保证对此变量的读写操作都不会被优化,每次中断发生,Main 都能够得到最新的 which_int。

- ✓ 多任务环境下各任务间共享的标志应该加volatile

  在多线程访问某字段时，我们希望这些访问能够读取到字段最新的值，同时写到变量的操作能够立即实现，声明字段的时候如果加上了volatile，那么对该字段的任何请求，包括读和写操作，都会立刻得到执行

- **Constant**

Constant 在英文字典里的意思是 “常数;恒量”,也就是说,把一个变量或对象定义成 Constant 类型,他的值是不能被更新的(read only)。定义以后不能被更新,那么在定义或说明常类型时必须给她一个初始值(行初始化)。
对于一些函数中的指针参数，如果在函数中只读，建议将其用 const 修饰

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

Const 指针的判断

如果 const 位于星号(*)的左侧，则 const 就是用来修饰指针所指向的变量，即指针指向为常量

const int *a

int const *a

因此，上面的情况相同，都是指针所指向的内容为常量（const 放在变量声明符的位置无关），这种情况下不允许对内容进行更改操作，

如不能*a = 3

如果 const 位于星号的右侧，const 就是修饰指针本身，即指针本身是常量。

int* const a

上面表示指针本身是常量，而指针所指向的内容不是常量，这种情况下不能对指针本身进行更改操作，如 a++是错误的

对于左右都有的情况如:

const int* const a

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

为指针本身和指向的内容均为常量。

- **Static**

一个 static 的变量，其实就是全局变量，只不过他是有作用域，她可用於保存变量所在函数被累次调用期间的中间状态,比如:

```
Int nCount()
{
   Static int cout=0;
   ……..
   Cout++;
    ……….
}
```

Cout 变量的值会跟随着函数的调用次而递增，函数退出后，Cout 的值还存在，只是 Cout 只能在函数中才能被访问(函数作用域)。而 Cout 的内存也只会在函数第一次被调用时才会被分配和初始化，以后每次进入函数，都不为 static 分配了，而直接使用上一次的值.

她还有一个作用就是访问控制:

- ✓ 在模块内（但在函数体外）一个被声明为静态的变量可以被模块内所用函数访问，但不能被模块外其它函

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

数访问。它是一个本地的全局变量。

✓ 在模块内，一个被声明为静态的函数只可被这一模块内的其它函数调用。那就是，这个函数被限制在声明它的模块的本地范围内使用。

如果别的模块中调用这个模块中的函数，或是使用其中的全局（用 extern 关键字），将会发生链接时错误。

- **_irq**

为了方便使用高级语言编写异常处理函数，ARM编译器对异常处理函数作了特定扩展，只要使用关键字__irq，这样编译出来的函数就满足异常响应对现场保护和恢复的需要.

关于编写中断服务程序的一些基本原则:

✓ **避免在中断服务程序中做浮点运算**

好的中断服务程序的应该遵循短而有效这一原则,但在中断服务程序中做浮点运算却大大地违背这一原则.同时有些处理器/编译器就是不允许在中断服务程序中做浮点运算

✓ **中断服务程序不能返回一个值**

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

所以中断服务程序都定义为返回类型为void,如:

void __irq Eint4567Isr(void)

✓ **中断服务程序不能传递参数**

所以中断服务程序的参数列表为void,如:

void __irq Eint4567Isr(void)

- **Unsigned 无符号类型变量定义,**

在C语言中,如果一个运算符两侧的操作数的数据类型不同，则系统按“先转换、后运算”的原则，首先将数据自动转换成同一类型，然后在同一类型数据间进行运算.

特别对无符号与有符号数据类型,有一个独特的自动转换规则往往被我们忽视.

比如定义下面两个变量:

unsigned int a=10;

signed int b=-100;

a>b?还是b>a?

实验证明b>a，也就是说－100>10，为什么会出现这样的结果呢?这是因为在C语言操作中，如果遇到无符号数与有符号数之间的操作，编译器会自动转化为无符号数来进行处理,因此a＝10，b＝4394985869，这样比较下去当然b>a了。

编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

● 访问绝对地址的内存位置

阅读上面的程序可以看到如下定义

#define pISR_EINT4567(*(unsigned *)(_ISR_STARTADDRESS+0x74))

她把无符号整数_ISR_STARTADDRESS+0x74 强制转换为指针,指向 RAM,我们用下面的方法访问她

注意到 Main 文件中如下语句

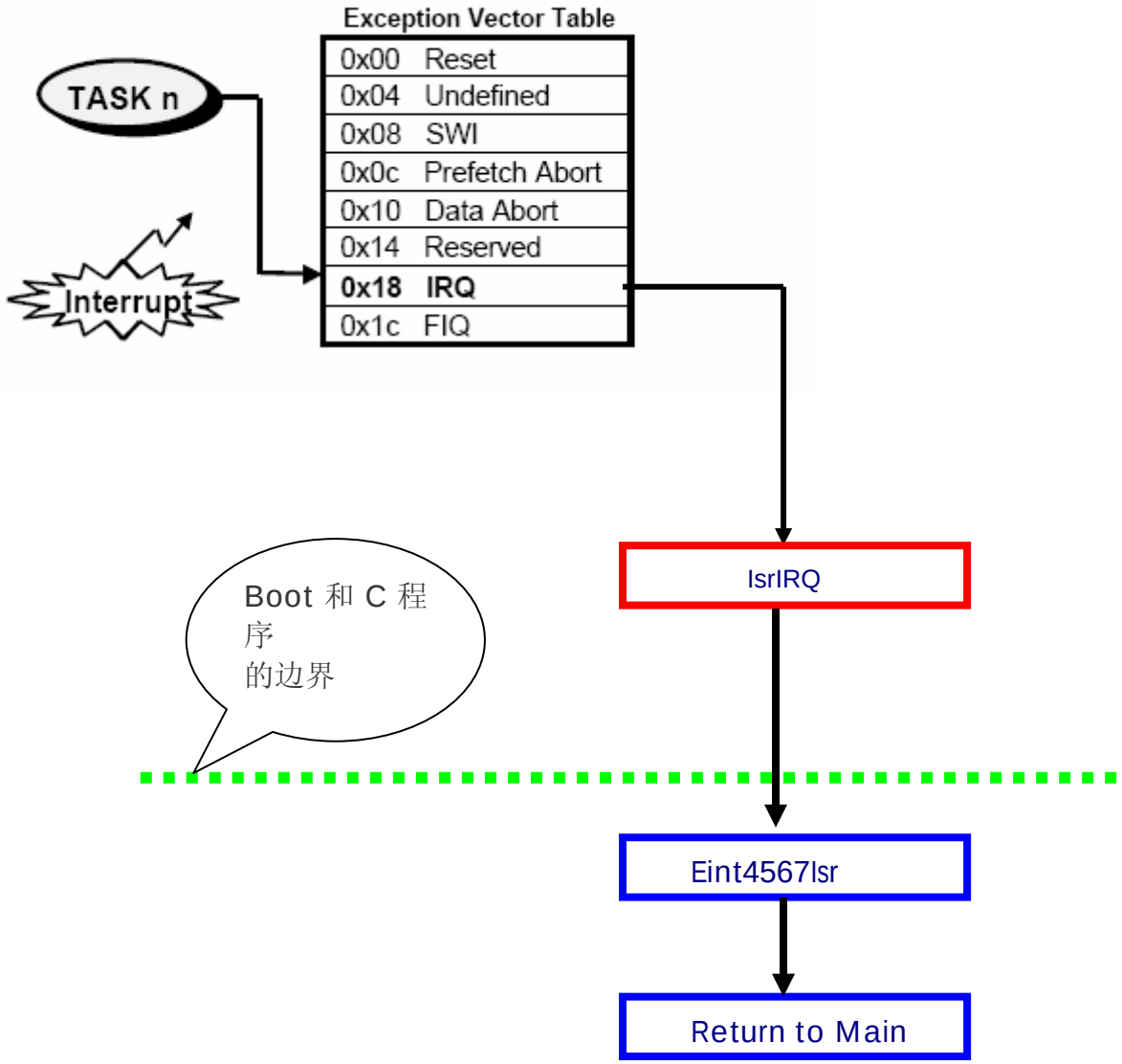
pISR_EINT4567=(int)Eint4567Isr;

就是这么简单:

- ✓ 为了访问一绝对地址,把一个整型数强制转换（typecast）为一指针

✧ 断服务程序运行流程图

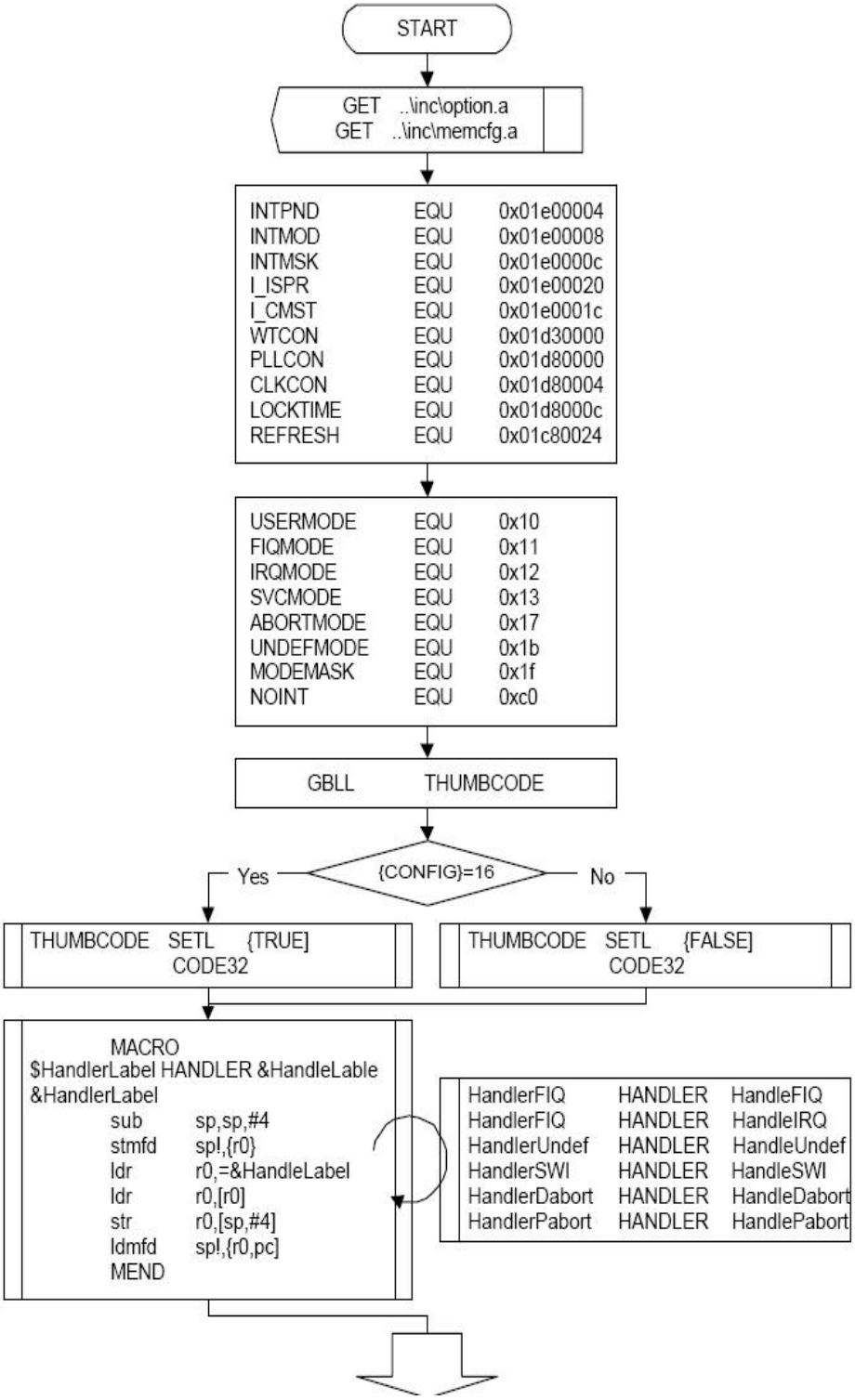
编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



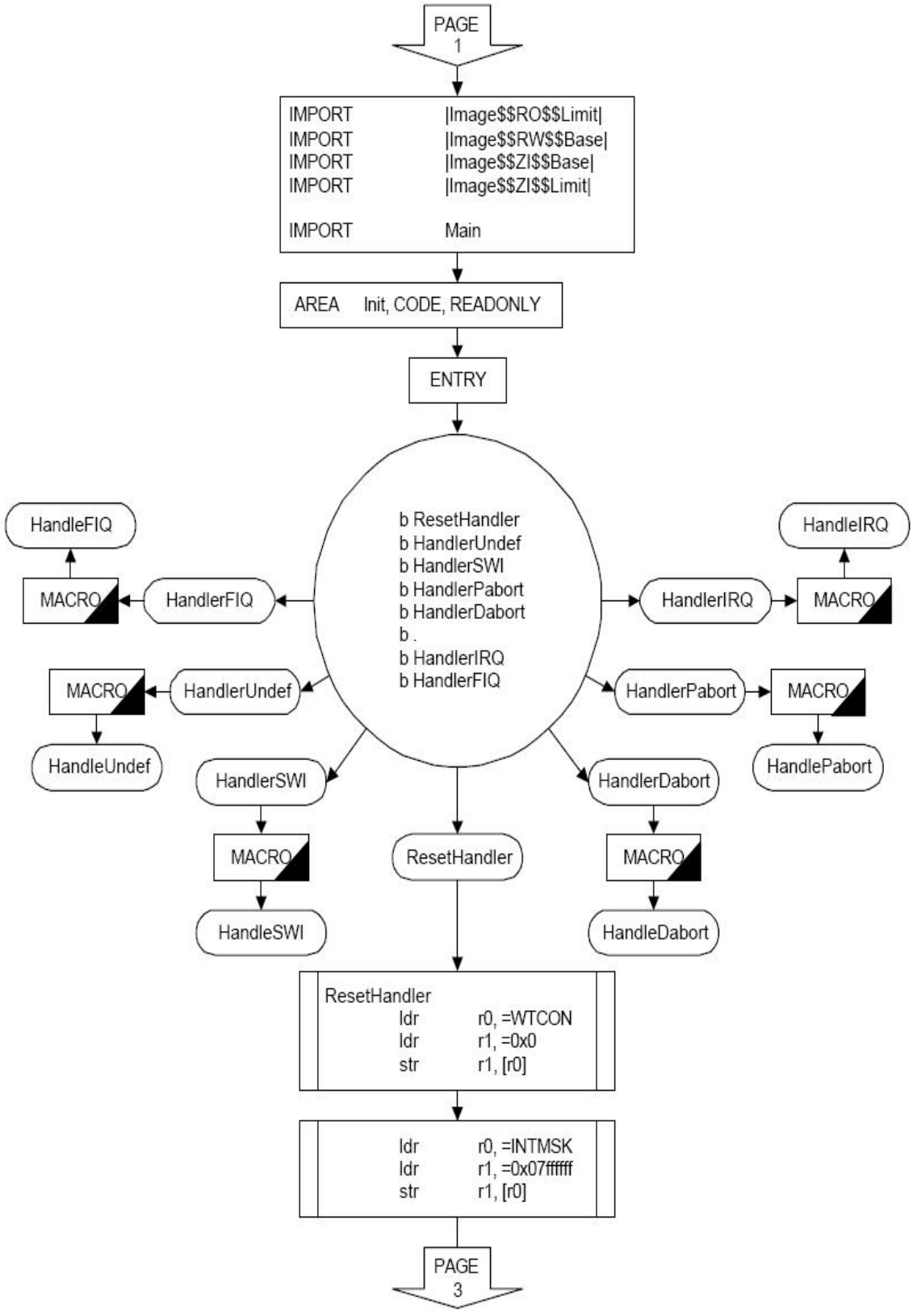
编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

第四章:ＢＯＯＴ流程图

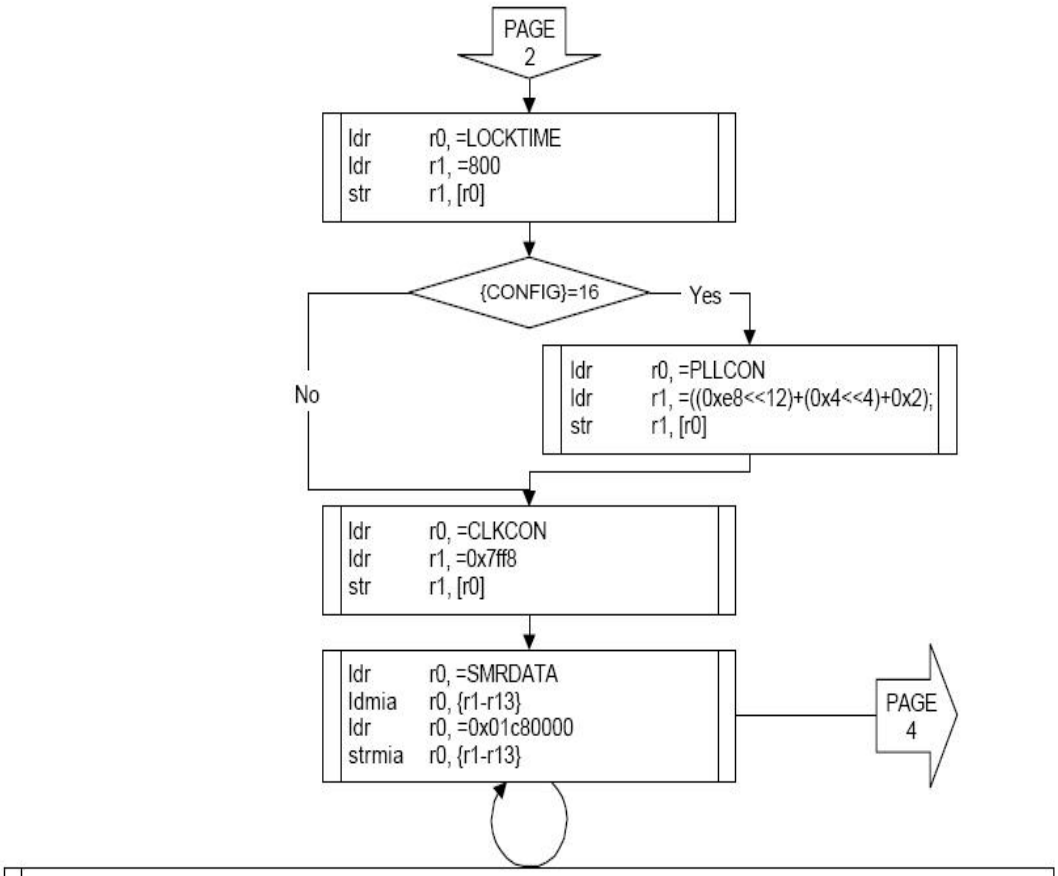
编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

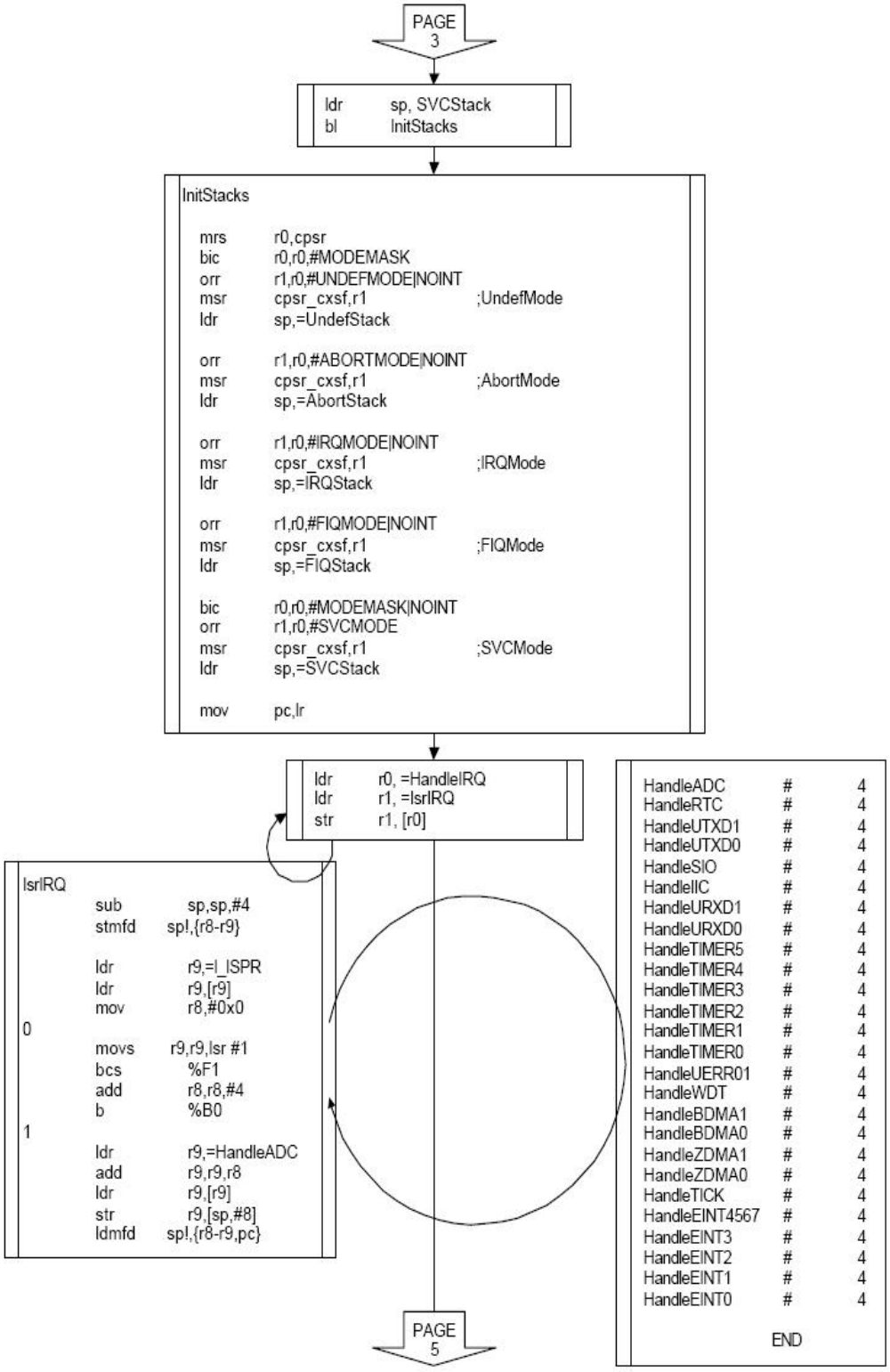


编者
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

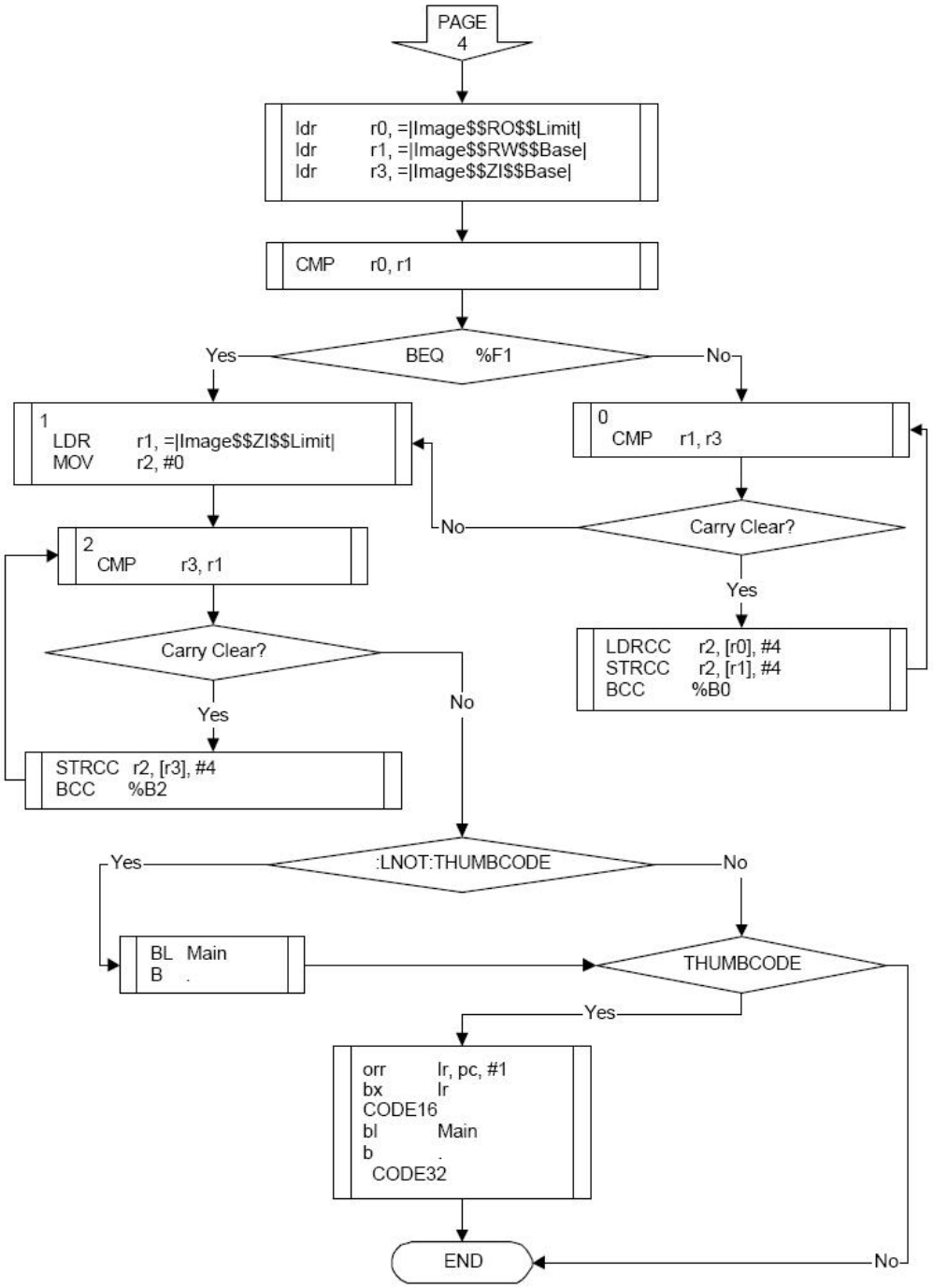



```
SMRDATA DATA
        [BUSWIDTH=16
                DCD 0x11111110
        |
                DCD 0x22222220
        ]
        DCD ((B0_Tacc<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC)
        DCD ((B1_Tacc<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC)
        DCD ((B2_Tacc<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC)
        DCD ((B3_Tacc<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC)
        DCD ((B4_Tacc<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC)
        DCD ((B5_Tacc<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC)
        [BDRAMTYPE="DRAM"
                DCD((B6_MT<<15)+(B6_Trcd<<4)+(B6_Tcas<<3)+(B6_Tcp<<2)+(B6_CAN))
                DCD((B7_MT<<15)+(B7_Trcd<<4)+(B7_Tcas<<3)+(B7_Tcp<<2)+(B7_CAN))
        |
                DCD((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
                DCD((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
        ]
        DCD ((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFCNT)
        DCD 0x10
        DCD 0x20
        DCD 0x20
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773



编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
EnterPWDN
        mov     r2, r0
        ldr     r0, =REFRESH
        ldr     r3,[r0]
        mov     r1, r3
        orr     r1, r1, #0x400000
        str     r1, [r0]

        nop
        nop
        nop
        nop
        nop
        nop
        nop

        ldr     r0, =CLKCON
        str     r2, [r0]

        mov     r0, #0xff
0       subs    r0, r0, #1
        bne     %B0

        ldr     r0, =REFRESH
        str     r3, [r0]
        mov     pc, lr

        LTORG
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

第 5 章

附录:BOOT程序源代码

```
; ********************************************************
; * NAME       : 44BINIT.S                                *
; * Version : 10.April.2000                               *
; * Description:                                          *
; * C start up codes                                      *
; * Configure memory, Initialize ISR ,stacks              *
; * Initialize C-variables                                *
; * Fill zeros into zero-initialized C-variables          *
; ********************************************************

        GET ..\inc\option.a
        GET ..\inc\memcfg.a

    ;Memory Area
    ;GCS6 64M 16bit(8MB) DRAM/SDRAM(0xc000000-0xc7fffff)
    ;APP       RAM=0xc000000~0xc7effff
    ;44BMON    RAM=0xc7f0000-0xc7fffff
    ;STACK       =0xc7ffa00
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;Interrupt Control
INTPND      EQU  0x01e00004
INTMOD      EQU  0x01e00008
INTMSK      EQU  0x01e0000c
I_ISPR      EQU 0x01e00020
I_CMST      EQU  0x01e0001c

;Watchdog timer
WTCON       EQU  0x01d30000

;Clock Controller
PLLCON      EQU  0x01d80000
CLKCON      EQU  0x01d80004
LOCKTIME    EQU  0x01d8000c

;Memory Controller
REFRESH     EQU 0x01c80024

;BDMA destination register
BDIDES0     EQU 0x1f80008
BDIDES1     EQU 0x1f80028

;Pre-defined constants
USERMODE    EQU   0x10
FIQMODE      EQU  0x11
IRQMODE      EQU  0x12
SVCMODE      EQU  0x13
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
ABORTMODE     EQU  0x17
UNDEFMODE     EQU  0x1b
MODEMASK      EQU  0x1f
NOINT         EQU  0xc0

;check if tasm.exe is used.
      GBLL    THUMBCODE
      [ {CONFIG} = 16
        THUMBCODE SETL  {TRUE}
      CODE32
      |
       THUMBCODE SETL  {FALSE}
      ]

      [ THUMBCODE
       CODE32     ;for start-up code for Thumb mode
      ]

      MACRO
$HandlerLabel HANDLER $HandleLabel

$HandlerLabel
      sub         sp,sp,#4
          ;decrement sp(to store jump address)
      stmfd     sp!,{r0}
          ;PUSH the work register to stack(lr does't push
          because it return to original address)
      ldr         r0,=$HandleLabel
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        ;load the address of HandleXXX to r0
ldr         r0,[r0]
        ;load the contents(service routine start address)
        of HandleXXX
str         r0,[sp,#4]
        ;store the contents(ISR) of HandleXXX to stack
ldmfd     sp!,{r0,pc}
         ;POP the work register and pc(jump to ISR)
MEND

IMPORT |Image$$RO$$Limit|
    ; End of ROM code (=start of ROM data)
IMPORT |Image$$RW$$Base|
    ; Base of RAM to initialise
IMPORT |Image$$ZI$$Base|
   ; Base and limit of area
IMPORT |Image$$ZI$$Limit|
   ; to zero initialise

IMPORT   Main       ; The main entry of mon program

AREA       Init,CODE,READONLY

ENTRY
b ResetHandler      ;for debug
b HandlerUndef      ;handlerUndef
b HandlerSWI        ;SWI interrupt handler
b HandlerPabort     ;handlerPAbort
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        b HandlerDabort    ;handlerDAbort
        b .                ;handlerReserved
        b HandlerIRQ
        b HandlerFIQ
      ;***IMPORTANT NOTE***
      ;If the H/W vectored interrutp mode is enabled, The
      above two instructions should
      ;be changed like below, to work-around with H/W bug
      of S3C44B0X interrupt controller.
       ; b HandlerIRQ   ->   subs pc,lr,#4
       ; b HandlerIRQ   ->   subs pc,lr,#4

    VECTOR_BRANCH
        ldr pc,=HandlerEINT0      ;mGA      H/W interrupt
vector table
        ldr pc,=HandlerEINT1      ;
        ldr pc,=HandlerEINT2      ;
        ldr pc,=HandlerEINT3      ;
        ldr pc,=HandlerEINT4567 ;
        ldr pc,=HandlerTICK       ;mGA
        b .
        b .
        ldr pc,=HandlerZDMA0      ;mGB
        ldr pc,=HandlerZDMA1      ;
        ldr pc,=HandlerBDMA0      ;
        ldr pc,=HandlerBDMA1      ;
        ldr pc,=HandlerWDT        ;
        ldr pc,=HandlerUERR01     ;mGB
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        b .
        b .
        ldr pc,=HandlerTIMER0     ;mGC
        ldr pc,=HandlerTIMER1     ;
        ldr pc,=HandlerTIMER2     ;
        ldr pc,=HandlerTIMER3     ;
        ldr pc,=HandlerTIMER4     ;
        ldr pc,=HandlerTIMER5     ;mGC
        b .
        b .
        ldr pc,=HandlerURXD0      ;mGD
        ldr pc,=HandlerURXD1      ;
        ldr pc,=HandlerIIC        ;
        ldr pc,=HandlerSIO        ;
        ldr pc,=HandlerUTXD0      ;
        ldr pc,=HandlerUTXD1      ;mGD
        b .
        b .
        ldr pc,=HandlerRTC        ;mGKA
        b .                       ;
        b .                       ;
        b .                       ;
        b .                       ;
        b .                       ;mGKA
        b .
        b .
        ldr pc,=HandlerADC    ;mGKB
        b .                       ;
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        b .                 ;
        b .                 ;
        b .                 ;
        b .                 ;mGKB
        b .
        b .
;0xe0=EnterPWDN
        ldr pc,=EnterPWDN

        LTORG

HandlerFIQ  HANDLER HandleFIQ
HandlerIRQ HANDLER HandleIRQ
HandlerUndef  HANDLER HandleUndef
HandlerSWI HANDLER HandleSWI
HandlerDabort  HANDLER HandleDabort
HandlerPabort HANDLER    HandlePabort

HandlerADC    HANDLER HandleADC
HandlerRTC  HANDLER HandleRTC
HandlerUTXD1  HANDLER HandleUTXD1
HandlerUTXD0  HANDLER HandleUTXD0
HandlerSIO  HANDLER HandleSIO
HandlerIIC  HANDLER HandleIIC
HandlerURXD1 HANDLER HandleURXD1
HandlerURXD0 HANDLER HandleURXD0
HandlerTIMER5 HANDLER HandleTIMER5
HandlerTIMER4 HANDLER HandleTIMER4
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
HandlerTIMER3 HANDLER HandleTIMER3
HandlerTIMER2 HANDLER HandleTIMER2
HandlerTIMER1 HANDLER HandleTIMER1
HandlerTIMER0 HANDLER HandleTIMER0
HandlerUERR01HANDLER HandleUERR01
HandlerWDT    HANDLER HandleWDT
HandlerBDMA1      HANDLER HandleBDMA1
HandlerBDMA0      HANDLER HandleBDMA0
HandlerZDMA1      HANDLER HandleZDMA1
HandlerZDMA0      HANDLER HandleZDMA0
HandlerTICK       HANDLER HandleTICK
HandlerEINT4567   HANDLER HandleEINT4567
HandlerEINT3    HANDLER HandleEINT3
HandlerEINT2    HANDLER HandleEINT2
HandlerEINT1    HANDLER HandleEINT1
HandlerEINT0    HANDLER HandleEINT0


;One of the following two routines can be used for
  non-vectored interrupt.

IsrIRQ ;using I_ISPR register.
        sub         sp,sp,#4            ;reserved for PC
        stmfd       sp!,{r8-r9}

     ;IMPORTANT CAUTION
      ;if I_ISPC isn't used properly, I_ISPR can be 0 in this
      routine.
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        ldr         r9,=I_ISPR
        ldr         r9,[r9]

    cmp       r9, #0x0   ;If the IDLE mode
                              work-around is used,
                         ;r9 may be 0 sometimes.
    beq       %F2

      mov       r8,#0x0
0
      movs       r9,r9,lsr #1
      bcs       %F1
      add       r8,r8,#4
      b       %B0

1
      ldr         r9,=HandleADC
      add       r9,r9,r8
      ldr         r9,[r9]
      str         r9,[sp,#8]
      ldmfd     sp!,{r8-r9,pc}

2
    ldmfd sp!,{r8-r9}
    add         sp,sp,#4
    subs    pc,lr,#4
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;*************************************************
;*  START                                        *
;*************************************************
ResetHandler
    ldr     r0,=WTCON       ;watch dog disable
    ldr     r1,=0x0
    str     r1,[r0]

    ldr     r0,=INTMSK
    ldr     r1,=0x07ffffff  ;all interrupt disable
    str     r1,[r0]

    ;*************************************************
    ;*Set clock control registers                    *
    ;*************************************************
    ldr   r0,=LOCKTIME
    ldr   r1,=0xfff
    str   r1,[r0]

    [ PLLONSTART
  ldr r0,=PLLCON        ;temporary setting of PLL
  ldr r1,=((M_DIV<<12)+(P_DIV<<4)+S_DIV)
                        ;Fin=10MHz,Fout=40MHz
  str  r1,[r0]
    ]

    ldr     r0,=CLKCON
    ldr     r1,=0x7ff8      ;All unit block CLK enable
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        str         r1,[r0]

        ;*************************************
        ;*   change BDMACON reset value for BDMA *
        ;*************************************
        ldr       r0,=BDIDES0
        ldr       r1,=0x40000000    ;BDIDESn reset value should
                                      be 0x40000000
        str       r1,[r0]

        ldr       r0,=BDIDES1
        ldr       r1,=0x40000000    ;BDIDESn reset value should
                                      be 0x40000000
        str       r1,[r0]

        ;*************************************************
        ;*Set memory control registers                   *
        ;*************************************************
        ldr         r0,=SMRDATA
        ldmia     r0,{r1-r13}
        ldr         r0,=0x01c80000   ;BWSCON Address
        stmia     r0,{r1-r13}

        ;*************************************************
        ;*Initialize stacks                              *
        ;*************************************************
        ldr         sp, =SVCStack;Why?
        bl          InitStacks
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;*****************************************************
;*Setup IRQ handler                                  *
;*****************************************************
ldr         r0,=HandleIRQ     ;This routine is needed
ldr           r1,=IsrIRQ          ;if there isn't 'subs
                                 pc,lr,#4' at 0x18, 0x1c
str         r1,[r0]

;*****************************************************
;*Copy and paste RW data/zero initialized data       *
;*****************************************************
LDR       r0, = |Image$$RO$$Limit |
          ; Get pointer to ROM data
LDR       r1, = |Image$$RW$$Base |  ; and RAM copy
LDR       r3, = |Image$$ZI$$Base |
;Zero init base => top of initialised data

CMP           r0, r1       ; Check that they are different
BEQ       %F1
0
CMP           r1, r3         ; Copy init data
LDRCC     r2, [r0], #4
     ;--> LDRCC r2, [r0] + ADD r0, r0, #4
STRCC     r2, [r1], #4
   ;--> STRCC r2, [r1] + ADD r1, r1, #4
BCC       %B0
1
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        LDR        r1, =|Image$$ZI$$Limit|
            ; Top of zero init segment
        MOV          r2, #0
2
        CMP          r3, r1        ; Zero init
        STRCC     r2, [r3], #4
        BCC       %B2

        [ :LNOT:THUMBCODE
          BL  Main          ;Don't use main() because ......
          B   .
        ]

        [ THUMBCODE         ;for start-up code for Thumb mode
          orr         lr,pc,#1
          bx           lr
      CODE16
          bl          Main        ;Don't use main() because ......
          b            .
      CODE32
          ]

;*****************************************************
;*  The function for initializing stack                *
;*****************************************************
InitStacks
    ;Don't use DRAM,such as stmfd,ldmfd......
    ;SVCstack is initialized before
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
;Under toolkit ver 2.50, 'msr cpsr,r1' can be used instead
of 'msr cpsr_cxsf,r1'

    mrs     r0,cpsr
    bic     r0,r0,#MODEMASK
    orr     r1,r0,#UNDEFMODE|NOINT
    msr     cpsr_cxsf,r1    ;UndefMode
    ldr     sp,=UndefStack

    orr     r1,r0,#ABORTMODE|NOINT
    msr     cpsr_cxsf,r1            ;AbortMode
    ldr     sp,=AbortStack

    orr     r1,r0,#IRQMODE|NOINT
    msr     cpsr_cxsf,r1            ;IRQMode
    ldr     sp,=IRQStack

    orr     r1,r0,#FIQMODE|NOINT
    msr     cpsr_cxsf,r1            ;FIQMode
    ldr     sp,=FIQStack

    bic     r0,r0,#MODEMASK|NOINT
    orr     r1,r0,#SVCMODE
    msr     cpsr_cxsf,r1            ;SVCMode
    ldr     sp,=SVCStack

  ;USER mode is not initialized.
    mov pc,lr ;The LR register may be not valid for the
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

mode changes.

```
;*************************************************
;*  The function for entering power down mode        *
;*************************************************
;void EnterPWDN(int CLKCON);
EnterPWDN
    mov     r2,r0                   ;r0=CLKCON
    ldr     r0,=REFRESH
    ldr     r3,[r0]
    mov     r1, r3
    orr     r1, r1, #0x400000       ;self-refresh enable
    str     r1, [r0]

    nop     ;Wait until self-refresh is issued. May not be
            needed.
    nop     ;If the other bus master holds the bus, ...
    nop     ; mov r0, r0
    nop
    nop
    nop
    nop

;enter POWERDN mode
    ldr     r0,=CLKCON
    str     r2,[r0]

;wait until enter SL_IDLE,STOP mode and until wake-up
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        ldr         r0,=0x10
0       subs        r0,r0,#1
        bne         %B0

    ;exit from DRAM/SDRAM self refresh mode.
        ldr         r0,=REFRESH
        str         r3,[r0]
        mov         pc,lr

        LTORG

    SMRDATA DATA
    ;*************************************************************
    ;* Memory configuration has to be optimized for best
      performance *
    ;* The following parameter is not optimized.
*
    ;*************************************************************

    ;*** memory access cycle parameter strategy ***
    ; 1) Even FP-DRAM, EDO setting has more late fetch point
        by half-clock
    ; 2) The memory settings,here, are made the safe
        parameters even at 66Mhz.
    ; 3) FP-DRAM Parameters:tRCD=3 for tRAC, tcas=2 for pad
         delay, tcp=2 for bus load.
    ; 4) DRAM refresh rate is for 40Mhz.
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
        [ BUSWIDTH=16
            DCD 0x11111110 ;Bank0=OM[1:0],
            Bank1~Bank7=16bit
        | ;BUSWIDTH=32
            DCD 0x22222220 ;Bank0=OM[1:0],
          Bank1~Bank7=32bit
        ]
      DCD
((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+
(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC))   ;GCS0
      DCD
((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+
(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC))   ;GCS1
      DCD
((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+
(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC))   ;GCS2
      DCD
((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+
(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC))   ;GCS3
      DCD
((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+
(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC))   ;GCS4
      DCD
((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+
(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC))   ;GCS5
      [ BDRAMTYPE="DRAM"
          DCD
((B6_MT<<15)+(B6_Trcd<<4)+(B6_Tcas<<3)+(B6_Tcp<<2)+(B6_
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
CAN))     ;GCS6 check the MT value in parameter.a
            DCD
((B7_MT<<15)+(B7_Trcd<<4)+(B7_Tcas<<3)+(B7_Tcp<<2)+(B7_
CAN))     ;GCS7
        | ;"SDRAM"
            DCD ((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
    ;GCS6
            DCD ((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
    ;GCS7
        ]
        DCD
((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)
+REFCNT)     ;REFRESH RFEN=1, TREFMD=0, trp=3clk, trc=5clk,
tchr=3clk,count=1019
        DCD 0x10            ;SCLK power down mode, BANKSIZE
32M/32M
        DCD 0x20            ;MRSR6 CL=2clk
        DCD 0x20            ;MRSR7

        ALIGN


        AREA RamData, DATA, READWRITE

        ^   (_ISR_STARTADDRESS-0x500)

    UserStack   #   256     ;c1(c7)ffa00
    SVCStack    #   256     ;c1(c7)ffb00
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
    UndefStack # 256    ;c1(c7)ffc00
    AbortStack # 256    ;c1(c7)ffd00
    IRQStack   # 256    ;c1(c7)ffe00
    FIQStack#  0    ;c1(c7)fff00


        ^   _ISR_STARTADDRESS
    HandleReset   #  4
    HandleUndef   #  4
    HandleSWI     #  4
    HandlePabort  #  4
    HandleDabort  #  4
    HandleReserved  #  4
    HandleIRQ     #  4
    HandleFIQ     #  4

    ;Don't use the label 'IntVectorTable',
    ;because armasm.exe cann't recognize this label
correctly.
    ;the value is different with an address you think it may be.
    ;IntVectorTable
    HandleADC     #  4
    HandleRTC     #  4
    HandleUTXD1   #  4
    HandleUTXD0   #  4
    HandleSIO     #  4
    HandleIIC     #  4
    HandleURXD1   #  4
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773

```
HandleURXD0  #  4
HandleTIMER5  #  4
HandleTIMER4  #  4
HandleTIMER3  #  4
HandleTIMER2  #  4
HandleTIMER1  #  4
HandleTIMER0  #  4
HandleUERR01 #  4
HandleWDT     #  4
HandleBDMA1 #  4
HandleBDMA0 #  4
HandleZDMA1 #  4
HandleZDMA0 #  4
HandleTICK     #  4
HandleEINT4567   #  4
HandleEINT3    #  4
HandleEINT2    #  4
HandleEINT1    #  4
HandleEINT0    #  4    ;0xc1(c7)fff84

        END
```


编者:王宇行(Nick)
Mail:Micro9229@yahoo.com
MSN:galaxy612@hotmail.com;
QQ: 86297143
Mobile:13661135773