(9 条消息) Arm 芯片上电启动流程剖解_Yannick_J 的博客 - CSDN 博客_arm 上电…

关键字：

stepping stone：(可以看做是 Internal RAM)

IROM (Internal ROM)：固化在 CPU 内部 ROM 里的一段代码，它的运行叫做 BL0.

IRAM： 因为 IROM 启动运行的时候，外置 SDRAM 还没有初始化好，而 IRAM 是可用的，因此必须要把 BL1 加载到 IRAM 中运行，由 BL1 对 SDRAM 进行初始化。ROM 为什么不初始化 SDRAM 呢？那是因为支持的 SDRAM 规格是可变的，由固化代码来初始化显得不够灵活，而且固化代码往往代码量比较小，因为越多越容易出 BUG，出 BUG 就会导致 SOC 芯片重新掩膜 tapout，一次可要好几百万人民币呢。

BL0:（BootLoad 0 阶段），BL0 做了些什么？

s5pc100 芯片手册见 2.2FUNCTIONAL SEQUENCE, 翻译成中文如下

1. 初始化 PLL 和时钟，将其设定为固定值；
2. 初始化栈和堆区域；

3. 初始化指令 Cache 控制器；

4. 从外部起动设备中加载 BL1；

5. 如果起动安全机制开启，则检查 BL1 数据完整性；

6. 如果校验通过，则跳转到 0x34010 地址处运行；

7. 如果校验失败则停止。

BL1: 从 CPU 上电起，把系统启动过程分为 3 个阶段 BL0、BL1、BL2。BL0 是固化在内部 ROM 上电就执行的一小段程序，BL0 引导 u-boot 的第一个阶段称为 BL1。通常加载到 CPU 内的 IRAM 中执行

BL2: 把 u-boot 的第二阶段代码用于引导内核的阶段称为 BL2，通常加载到 SDRAM 中执行。

谈到 arm 的启动流程不得不说的是 bootloader，但是我这篇文章主要来谈谈 arm 启动流程的，所以 bootloader 只是跟大家简介一下就 ok。这篇文章我会谈到以下内容：

1、bootloader 简介以及其作用

2、2440、6410、210 当下比较常见的 3 款处理器的启动流程进行简单分析，通过这三款处理器的分析希望大家掌握 arm 处理器的启动分析。

Ok 我们进入主题

l Bootloader 简介及其作用

在我看来 bootloader 的作用是初始化必要的硬件，引导内核启动。（当然这是主要作用，今天的重点不在 bootloader，所以在我后面的博文会继续谈到的）

l 启动流程分析

在分析启动流程的时候我们将会使用的文档是三星公司提供的芯片手册，通过手册我们搞清楚芯片的启动。

在分析启动流程之前我们首先要清楚不论是 arm 的何种处理器，其都是从 0x0000 0000 地址处开始执行程序的。下面的分析我将会通过三个方面：
1、芯片支持的启动方式 2、地址布局 3、启动流程

1. 2440

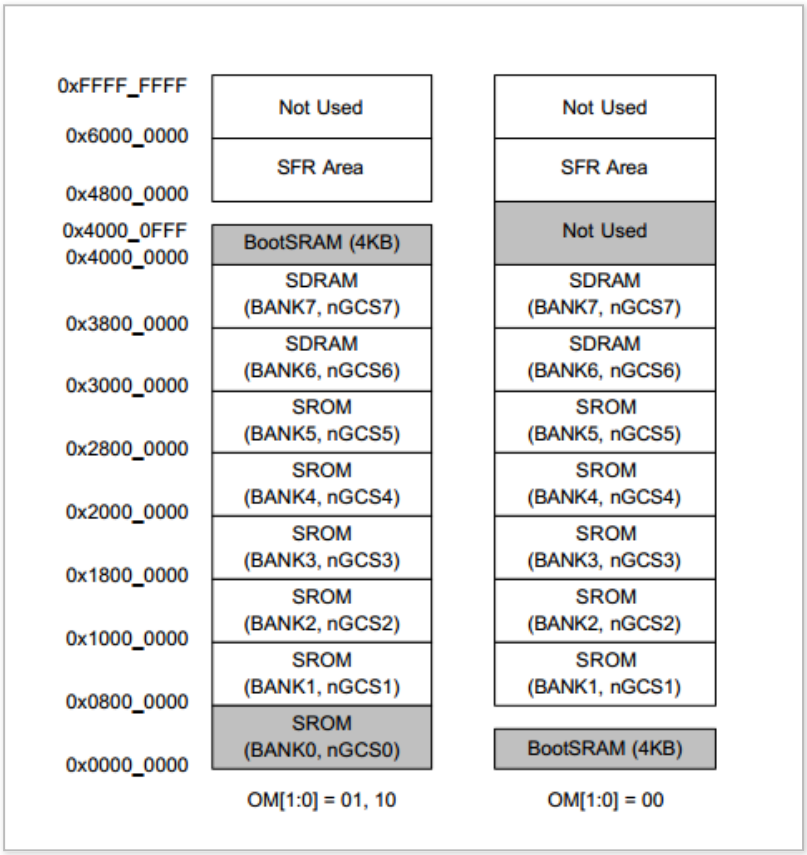
a) 启动方式

由上图可知，S3C2440 支持两种启动模式：NAND 和非 NAND（这里是 Nor Flash），具体采用的方式取决于 OM0、OM1 两个引脚的状态。

OM1 (Operating Mode 1)	OM0 (Operating Mode 0)	Booting ROM Data width
0	0	Nand Flash Mode
0	1	16-bit
1	0	32-bit
1	1	Test Mode

b) 地址布局

我们知道 arm 从 0 地址出运行代码那么我们的零地址处到底存放的是什么东西呢？我们通过地址布局图来分析



从上图我们可以清楚的看到左边的是从 Nor Flash 启动的地址布局，右边是从 NAND 启动的地址布局，因为 Nor Flash 内可以运行程序 (Nor flash 地址线和数据线独立，nand flash 地址线 / 数据线复用)，所以我们在放 bootloader 的时候放在 0 地址处即可，所以我们重点分析从 NAND 启动。

c) 启动流程

我们从地址布局图中可以看到，当我们从 NAND 启动的时候 0 地址处是 BootSRAM（又叫做 stepping stone 垫脚石），当我们上电时其会做以下事情

上电后处理器自动将 nandflash 前 4KB 的内容复制到 boot sram(I RAM，CPU 内部 RAM) 开始执行，这一过程就是 BL1 加载过程（由 CPU 内部的 BL0 代码程序完成）。

通过 bootsram（即刚才复制进来的 4k）来初始化相关硬件和寄存器从而访问 nandflash，接下来把剩余的 bootloader 复制到内存（SDRAM/DRAM）中（即 BL1->BL2），当 stepping stone 里面的 4KB 执行完以后跳转到内存继续执行，完成系统的启动。

2. 6410

a) 启动方式

Table 3-1. Device operating mode selection at boot-up

XSELNAND	OM[4:0]	GPN[15:13]	Boot Device	Function	Clock Source
1	0000X	XXX	RESERVED	RESERVED	XXTIpll if OM[0] is 0. XEXTCLK if OM[0] is 1
1	0001X			RESERVED	
1	0010X			RESERVED	
1	0011X			RESERVED	
X	0100X		SROM(8bit)	-	
X	0101X		SROM(16bit)	-	
0	0110X		OneNAND2	Don't use Xm0CSn2 for SROMC	
X	0111X		MODEM	Don't use Xm0CSn2 for SROMC	
X	1000	1111X	IROM	SD/MMC(CH0)	
0	001			OneNAND	
1	010			NAND(512Byte, 3-Cycle)	
1	011			NAND(2048Byte, 4-Cycle)	
1	100			NAND(2048Byte, 4-Cycle)	
1	101			NAND(2048Byte, 5-Cycle)	
1	110			NAND(4096Byte, 5-Cycle)	
X	111			SD/MMC(CH1)	

从上图我们可以看到 6410 支持的启动方式比较多有 SROM（Nor Flash）启动，oneNAND 启动，IROM；即内置 ROM（IROM 是处理器内部的固件 / 存储器，但不是 stepping stone）启动。其中 IROM 中又有 sd 卡、NAND，我们可以配置相应的管脚去选择其启动方式。

b) 地址布局

Table 2-2. Device Specific Address Space

Address		Size(MB)	Description	Note
0x0000_0000	0x07FF_FFFF	128MB	Booting Device Region by XOM Setting	Mirrored Region
0x0800_0000	0x0BFF_FFFF	64MB	Internal ROM	
0x0C00_0000	0x0FFF_FFFF	64MB	Stepping Stone (Boot Loader)	
0x1000_0000	0x17FF_FFFF	128MB	SROMC Bank0	
0x1800_0000	0x1FFF_FFFF	128MB	SROMC Bank 1	
0x2000_0000	0x27FF_FFFF	128MB	SROMC Bank 2	
0x2800_0000	0x2FFF_FFFF	128MB	SROMC Bank 3	
0x3000_0000	0x37FF_FFFF	128MB	SROMC Bank 4	
0x3800_0000	0x3FFF_FFFF	128MB	SROMC Bank 5	
0x4000_0000	0x47FF_FFFF	128MB	Reserved	
0x4800_0000	0x4FFF_FFFF	128MB		
0x5000_0000	0x5FFF_FFFF	256MB	DRAM Controller of the Memory Port1	
0x6000_0000	0x6FFF_FFFF	256MB		

从上图我们可以发现在 0 地址处是一个镜像区，不放置任何设备，当我们选择不同的启动方式的时候，其通过映射关系将对应的设备映射到镜像区域。比如我们选择从 IROM 启动其就会将 IROM 映射到该区域。

c) 启动流程

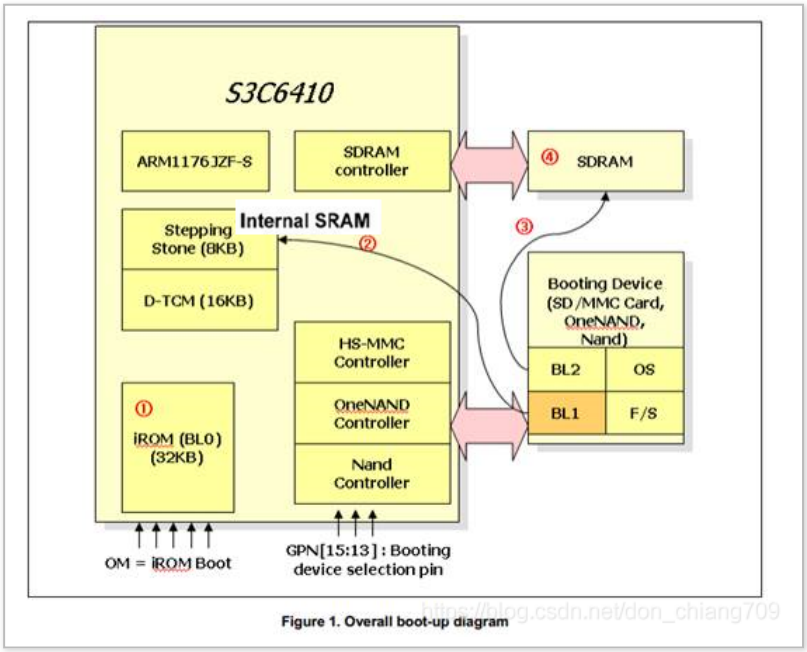


Figure 1. Overall boot-up diagram

假设我们从 NAND 启动，从启动方式中我们可以知道从 NAND 启动是属于从 IROM 启动的，所以当我们上电的时候其会做以下事情：

1. 将 IROM 映射到镜像区

2. IROM 中有芯片厂商写好的 BL0，由 BL0 将系统引导至启动选项，然后将 BL1（NAND 中前 8k）拷进 stepping stone（IRAM，即内置 RAM）进行运行

3. BL1 将剩下的 BL2 拷进内存，当 BL1 执行完以后跳转到内存继续执行，完成系统的启动。

```
D-TCM==数据紧密耦合内存 (Data Tightly Coupled Memory)。
TCM 是一段始终有效的连续内存区域（如果启用了 TCM）。 TCM 用作
TCM 用于向处理器提供低延迟内存，它没有高速缓存特有的不可预测性。
有关 TCM 的完整体系结构描述，请参阅《ARM 体系结构参考手册》以及
参考：https://blog.csdn.net/otianshizairenjian/article
```

对 ARM 紧致内存 TCM 的理解

3. S5PV210

S5PV210 属于这一类系统，外挂 SDRAM 内存和 NAND、SD 卡等外存设备，系统和应用代码等作为固件存放在外存设备中，并通过 S5PV210 内置的 IROM 启动并逐步引导到 SDRAM 中。S5PV210 的 SPEC 是《S5PV210_UM_REV1.1.pdf》，在其第 29 页~ 30 页中描述了 IROM 和 IRAM 的内存映射图。IRAM，即内置 RAM，在启动引导阶段有两个作用：一是 IROM 运行时使用的数据变量所在的区域，二是 IROM，即 BL0 会将外存中的 BL1 引导到该区域中。因为 IROM 启动运行的时候，外置 SDRAM 还没有初始化好，而 IRAM 是可用的，因此必须要把 BL1 加载到 IRAM 中运行，由 BL1 对 SDRAM 进行初始化。IROM 为什么不初始化 SDRAM 呢？那是因为支持的 SDRAM 规格是可变的，由固化代码来初始化显得不够灵活，而且固化代码往往代码量比较小，因为越多越容易出 BUG，

出 BUG 就会导致 SOC 芯片重新掩膜 tapout，一次可要好几百万人民币呢。由内存映射图可以得到：

IROM 是 64K，在 0x0000-0000 开始的区域，而 IRAM 是 96K，在 0xD0020000-0xD0037FFF。我们可以看到其地址布局和 6410 类似，采用了映射的方法。S5PV210 的 IROM 的 SPEC 是《S5PV210_iROM_ApplicationNote_Preliminary.pdf》，其主要描述以下内容

1. 启动流程，分析如下：

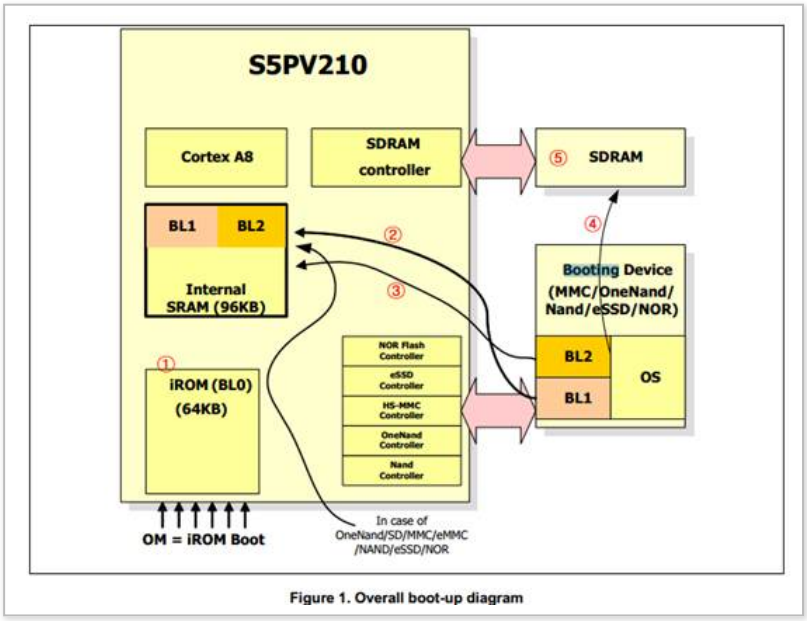


Figure 1. Overall boot-up diagram

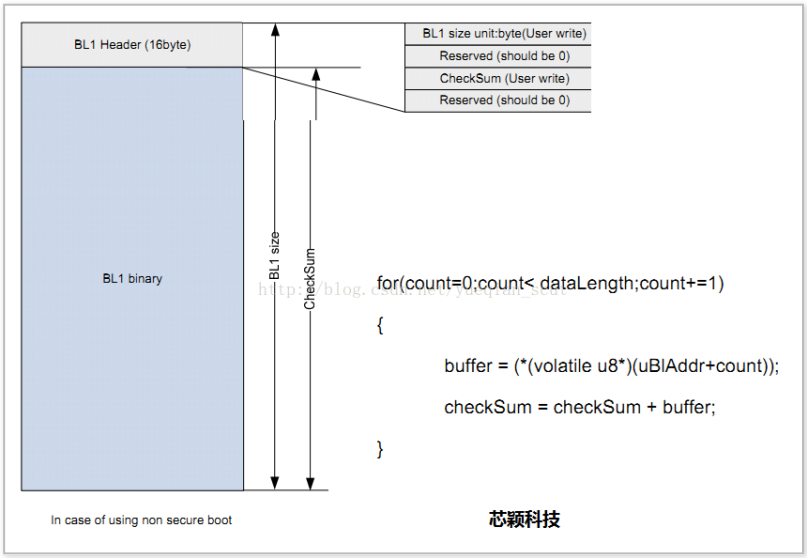
1）BL0 位于 IROM 中，其选择引导的介质受外围引脚 OM 电平所决定。

其能引导的介质包括：nor，nand，onenand，SD/MMC 等等。如下图，0x2 表示 nand（page 为 2k byte，5 个 command 周期，8bit ECC）启动，

0xc 表示 SD/MMC。OM 的值可以通过读寄存器 0xE000-0004 得到。

OM[5]	OM[4]	OM[3]	OM[2]	OM[1]	OM[0]	OM[5]	OM[4]	OM[3]	OM[2]	OM[1]	OM[0]
1'b0	1'b0	1'b0	1'b0	1'b0	1'b0	Boot Mode	I-ROM	eSSD			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			Nand 2KB, 5cycle (Nand 8bit ECC)			X-TAL
					1'b1						X-TAL(USB)
			1'b1	1'b0	1'b0			Nand 4KB, 5cycle (Nand 8bit ECC)			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			Nand 4KB, 5cycle (Nand 16bit ECC)			X-TAL
					1'b1						X-TAL(USB)
		1'b1	1'b0	1'b0	1'b0			OnenandMux(Audi)			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			OnenandDemux(Audi)			X-TAL
					1'b1						X-TAL(USB)
			1'b1	1'b0	1'b0			SD/MMC			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			eMMC(4-bit)			X-TAL
					1'b1						X-TAL(USB)
	1'b1	1'b0	1'b0	1'b1	1'b0			Nand 2KB, 5cycle (16-bit bus, 4-bit ECC)			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			Nand 2KB, 4cycle (Nand 8bit ECC)			X-TAL
					1'b1						X-TAL(USB)
			1'b1	1'b0	1'b0			iROM NOR boot			X-TAL
					1'b1						X-TAL(USB)
				1'b1	1'b0			eMMC(8-bit)			X-TAL
					1'b1						X-TAL(USB)
芯颖科技			1'b0	1'b0	1'b0			eSSD			X-TAL
					1'b1						X-TAL(USB)

2）根据 OM 的值，IROM 中的 BL0 选择其固话的对应驱动将对应介质的前 16K 代码数据读到 IRAM 中。引导代码之后会进行校验，校验和 BL1 的长度信息放在 16K 代码数据的最前面 16 个字节。为什么需要长度？因为校验是对确定长度的内容进行计算得到，而 BL1 的有效代码数据可能并没有 16K。BL1 头部信息和校验算法如下图：



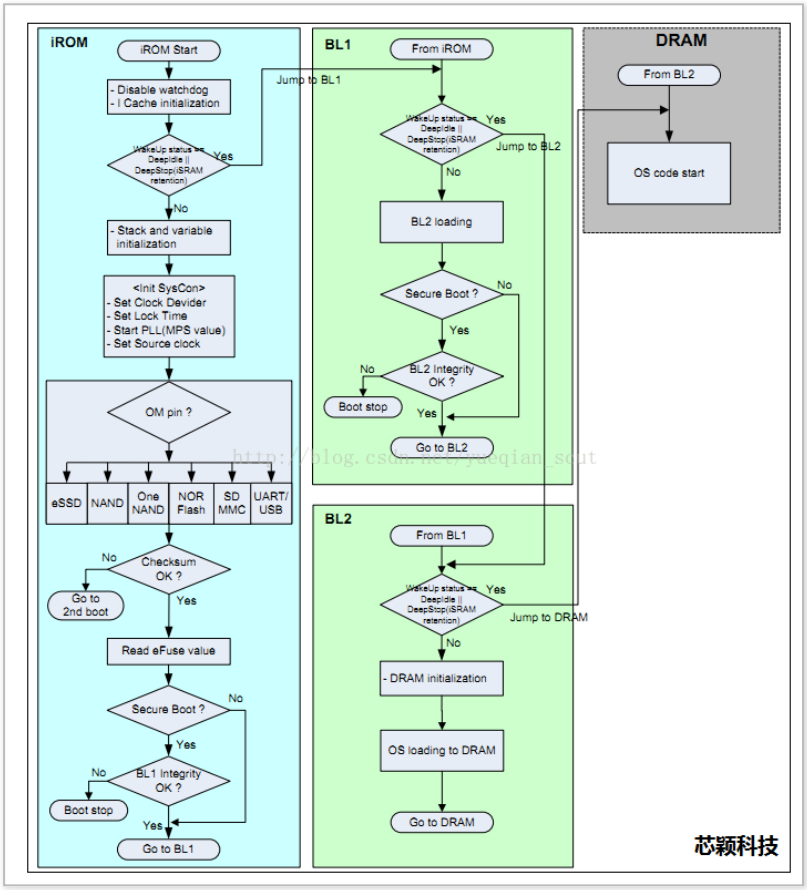
加载 BL1 之后，BL0 会将给 PC 赋值为 **0xd0020010**，即 BL1 代码真正的入口应该设置链接到 0xd0020010。在编译链接生存 BL1 之后需要用专门的工具计算出 BL1 的长度和校验码，并填充到 BL1 的头部，在 uboot 中称为 NAND_SPL.BIN。

3）启动过程中的示意图的第三步只是一种方案的建议，在实际的启动模块中，BL2 一般比较大，因为其包括引导操作系统，还包括在启动阶段支持下载等交互功能，所以 BL2 会大于 IRAM 中剩下的 80K，因此，BL1 执行时会先初始化好 SDRAM，然后将 BL2 引导到 SDRAM。因此图中的第三步示意图并不准确。BL1 执行时 MMU 是关闭的，其在建立好临时页表后，会开启 MMU，并跳转到 BL2 开始执行。

4）BL2 会提供与用户交互的模式命令，一般用于研发人员调试，如果是真正的产品则没有这种模

式，直接开始引导 OS 到 SDRAM。并跳转到 OS 开始执行。

5）一般的启动流程图如下（这个图很赞，可以仔细看看）：



从上图可以得知，当引导 OM 所对应的介质的 BL1 失败（如介质接口有问题、校验失败等）

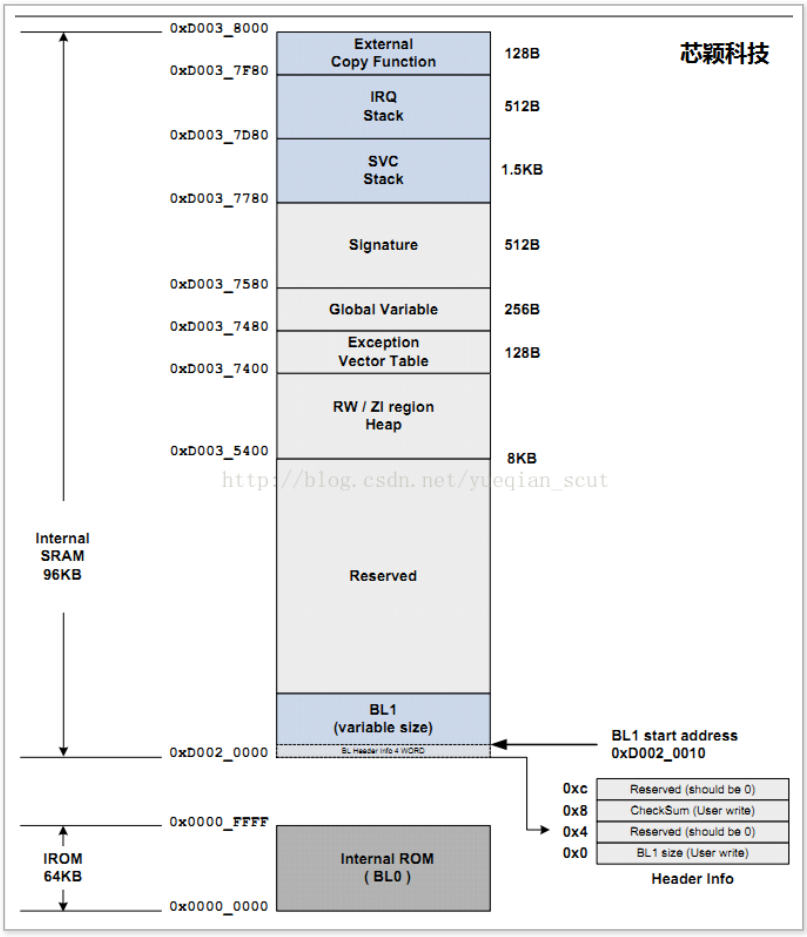
时，还有另外的备选引导方案，即 2nd boot，那就是 USB 引导和串口引导，这里不再展开。

2. IROM 除了引导之后，还做了一些硬件和软件环境初始化工作。主要流程包括：

1）关闭看门狗

2）初始化指令 cache，开启指令 cache

3）初始化 IRAM 中的栈（中断栈、SVC 栈）和堆，异常向量注册表等。具体如下图：



IROM 运行是处于 SVC 模式。这里的 exception vector table 并不是异常向量表，而应该称为异常向量注册表。因为异常向量表是在 IROM 的起始位置上，即 0x00000000 开始的地方，如下所示，IROM 建立起的环境只能支持中断（异常的一种）注册，即注册到 0xd0037418，对于其他异常的支持需要在启动或者 OS 阶段进行重建，并将其定位到 SDRAM 区域，并设置异常向量的基地址（通过改变协处理器 P15 的 C12-Vector Base Address Register）指向该区域。

```
_start: b reset
        .word 0x0
        .word 0x0

        .word 0x0

        .word 0x0

        .word 0x0

        ldr pc, _irq
        .word 0x0
```

4）填充好 IRAM 的块设备读接口区域。如上图，0xd0037f80 开始的 128 个字节存放的是块设备的拷贝函数地址，如 nand，sd/mmc 等的读

接口地址。其用于后续 BL1 和 BL2 使用。即 BL1 和 BL2 代码不需要实现外存设备的读驱动函数，只需要调用 IROM 的接口就可以了。SPEC 的 p14-p18 具体描述了各种不同的介质设备的操作接口地址以及相关的信息。

5）开启 PLL，初始化系统时钟。

6）copy 16k 字节的 BL1 到 IRAM。

7）对 BL1 进行校验，即计算 BL1 的校验值，并与 BL1 的头部的校验值进行比较，失败即会跳到 2nd boot 中去引导，成功即跳到 BL1 的 0xd0020010 执行。

转载：

Arm 启动流程解析

ARM 处理器启动分析（BL0/BL1/BL2）

扩展：

系统引导时为什么要关闭 I/D Caches？

ARM-I/Dcache, MMU 关系

Arm I/D cache 研究

ARM 官网支持

全文完