Application Report
ZHCA125 – November 2011

DM368 NAND Flash 启动揭秘

孟海燕 TI 通用数字信号处理系统技术支持

摘要

芯片上电后是如何启动实现应用功能的？这是许多工程师在看到处理器运行的时候，通常都会问的一个问题。本文以德州仪器的多媒体处理芯片 TMS320DM368 为例，介绍它的 NAND Flash 启动原理。并且以 DM368 IPNC 参考设计软件为例，介绍软件是如何配合硬件实现启动的。

内容

图

表

1 NAND Flash 启动原理

德州仪器的多媒体处理芯片 TMS320DM368 可以实现 1080P30 h264 的编码，已经广泛的使用在了网络摄像机的应用中。DM368 可以支持 NOR Flash, NAND Flash, UART, SD Card 启动等多种启动方式。

1.1 DM365 支持的 NAND 启动特性

1. 不支持一次性全部固件下载启动。相反的，需要使用从 NAND flash 把第二级启动代码（UBL）复制到 ARM 的内存（AIM），将控制转交给用户定义的 UBL。
2. 支持最大 4KB 页大小的 NAND。
3. 支持特殊数字标志的错误检测，在加载 UBL 的时候会尝试最多 24 次。例如在 NAND 的第 1 个 block 没有找到特殊数字标志，会到下一个 block 继续查找，一直到查找到第 24 个 block。

4. 支持 30KB 大小的 UBL（DM365 有 32KB 的内存，其中 2KB 用作了 RBL 的堆栈，剩下的空间可以放 UBL ）
5. 用户可以选择在 RBL 执行的时候是否需要支持 DMA，I-cache（例如，加载 UBL 的时候）
6. 使用并且需要 4 位硬件 ECC（支持每 512 字节需要 ECC 位数小于或等于 4 位的 NAND Flash）。
7. 支持需要片选信号在 Tr 读时间为低电平的 NAND Flash。

1.2 NAND Flash 启动流程

在网络网络摄像机的应用中为了节约成本，有一些用户使用了 NAND Flash 启动方式。图 1 就是从上电到 Linux 启动的一个概要的流程图。首先 RBL（ROM boot loader）从 NAND 上读取 UBL（user boot loader）并且复制到 ARM 的内存里面。UBL 运行在 ARM 的内存里，初始化系统，例如初始化 DDR。然后 UBL 从 NAND Flash 里面读取 U-Boot 的内容并且复制到 DDR 里运行。DDR 里面运行的 U-Boot 又从 NAND Flash 里面读取 Linux 内核代码，并且复制到 DDR 上，然后启动内核。这样 DM365 的系统就从上电到完成 Linux 内核启动，然后就可以运行相应的应用程序了。

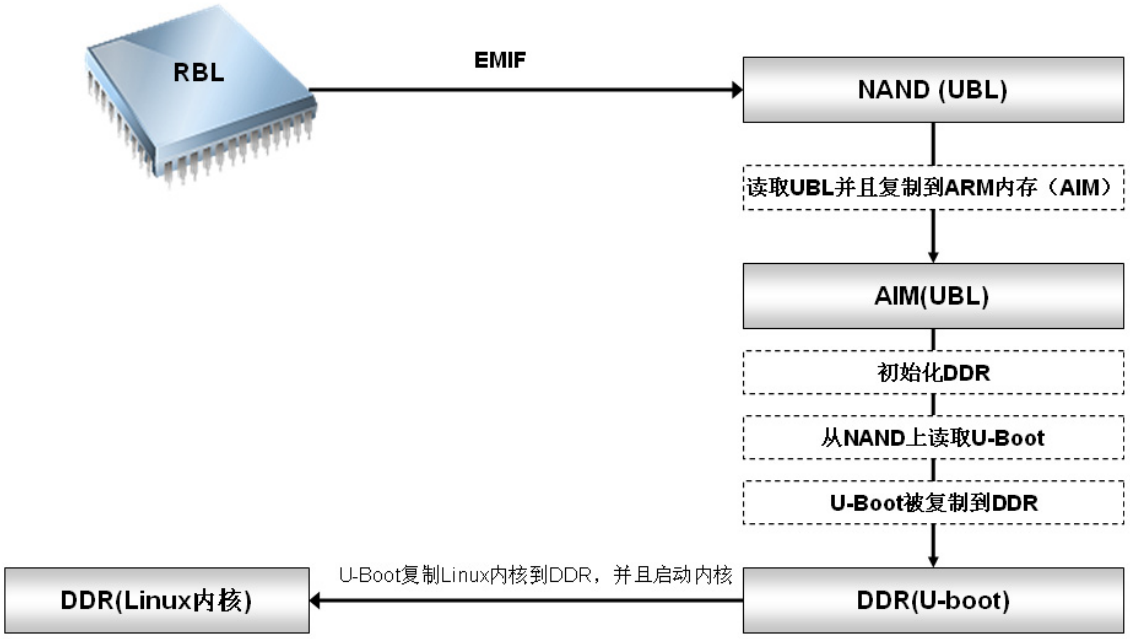

图 1. NAND Flash 启动流程图

下面我们会一步一步的介绍从上电到 Linux 启动是如何实现的。

首先我们需要提到的一个概念是 RBL，也就是 ROM Boot Loader （ROM 启动代码）。在 DM368 芯片上有一块 ROM 的区域（地址从 0x00008000 到 0x0000 BFFF），这块区域就是存放 RBL 代码的地方。ROM 上的代码是在芯片出厂前就烧写好的，用户是不能修改的。在 DM368 上，除了 AEMIF （Nor Flash）启动，其他的启动方式都需要运行 RBL。

无论是上电复位，热复位，还是看门狗复位，在复位信号由低到高的时候，DM368 芯片会检测 BTSEL[2:0]引脚（启动选择引脚）。只要检测到电平不是 001，也就是不是 AEMIF （NOR Flash）启动，ARM 程序就会从 ARM 的 ROM 的地址 0x00008000 地址开始执行。

RBL 首先会读取 BOOTCFG 寄存器里面的 BTSEL 信息，如果发现 BTSEL 的状态是 000，就会得知配置的是 NAND Flash 启动，NAND 启动模式开始执行。注意：为了保证 NAND 启动正常运行，需要保证在复位的时候 DEEPSLEEPZ/GIO0 引脚拉高。在确认启动是 NAND 后，首先 RBL 会初始化最高 2KB 的内存为堆栈并且关闭所有中断。然后 RBL 会读取 NAND 的 ID 信息，然后在 RBL 的代码里面的 NAND ID 列表，从而得知更详细的 NAND Flash 的信息，例如页（page）大小等，对 EMIF 做好相应的配置。DM368 支持启动的 NAND 的 ID 信息可以在参

考文档 1（ARM 子系统用户手册）里面找到。硬件选型时，请务必选择在 NAND ID 列表里面支持的 NAND 芯片。

接下来，RBL 会在 NAND Flash 的第 1 块的第 0 个页开始查找 UBL 的描述符。如果没有找一个合法的 UBL 的特殊数字标志，RBL 会继续到下一个块的第 0 个页查找描述符，最多第 24 个块。RBL 会到多个块里面查找描述符是根据 NAND Flash 本身容易有坏块的特点而设计的。24 块应该足以避免 NAND Flash 坏块的影响。

如 RBL 在某块里面找到了合法的 UBL 描述符，这个块号（block number）就会写到 ARM 内存最后的 32 位（0x7FFC~0x8000）用于调试时候使用，然后 UBL 描述符的具体内容将被读取并且处理。UBL 描述符告诉 RBL 关于下载和将控制权交给 UBL 所需要的信息，具体见表 1.

第 0 页地址	32 位	描述
0	0xA1AC Edxx	特殊数字标志
4	UBL 入口地址	用户启动代码的入口地址（绝对地址）
8	UBL 使用的页数	页数（用户启动 diamond 的大小）
12	UBL 开始的块号	开始存放用户启动代码的块号（block number）
16	UBL 开始的页数	开始存放用户启动代码的页数
20	PLL 设置-M	PLL 设置- 倍率（仅在特殊数字标准表示 PLL 使能的时候有效）
24	PLL 设置-N	PLL 设置- 分率（仅在特殊数字标准表示 PLL 使能的时候有效
28	快速 EMIF 设置	快速 EMIF 设置（仅在特殊数字标准表示快速 EMIF 启动的时候有效）

表 1. NAND UBL 描述符

一旦用户需要的启动设置配置好，RBL 就会从 0x0020 地址开始把 UBL 搬移到 ARM 内存。在从 NAND 读取 UBL 的过程中中，RBL 会使用 4 位的硬件 ECC 对 NAND Flash 上的数据进行检错和纠错。如果因为其他原因读失败，复制会立即停止，RBL 会在下个块里面继续寻找特殊数字标志。

对于 UBL 的描述符有几点注意事项：

1. 入口地址必须在 0x0020 到 0x781C 之间
2. 存放 UBL 的页必须是连续的页，可以分布在多个块内，总共大小必须小于 30KB。
3. UBL 的起始块号（block number）可以是和存放 UBL 描述符的块号一样。
4. 如果 UBL 的起始块号是和存放 UBL 描述符的块号一样， 那 UBL 的起始页数一定不可以和 UBL 描述符存放的页数一样。

但 RBL 根据 UBL 描述符里提供的 UBL 大小信息将 UBL 全部成功复制到 ARM 内存后，RBL 会跳到 UBL 起始地址，这样芯片的控制权就交给了 UBL，UBL 开始在 ARM 内存里运行了。

也许你会问，既然 RBL 可以把 NAND Flash 上的内容复制到 ARM 内存里运行，为什么我们不直接把 U-Boot 复制到内存运行？原因是 ARM 内存太小。一般的 U-Boot 都是大于 100KB，而 DM365 上可以用于启动的内存只有 30KB。也许你又要问了，那为什么不把 U-Boot 直接复制到 DDR 上运行，DDR 有足够大的空间？这个原因是，芯片上电后并无法知道用户在 DM365 的 DDR2 接口上接的 DDR 信息，RBL 也就无法初始化 DDR，在 RBL 运行的阶段 DDR 是不可用的。这也是为什么 UBL 里面初始化 DDR 是它的一项重要任务。

当 NAND 启动失败的时候，RBL 会继续尝试 MMC/SD 启动方式。如果你系统使用 NAND 启动，但 NAND 上的内容损坏了，如果你的板子上有 SD 卡接口，不需要改变启动方式，你就可以用 SD 卡先把系统启动起来，然后重新烧写 NAND Flash 上的内容。这样板子又可以正常工作了。这可以作为产品失效后在客户侧的一个补救方法。

2 NAND Flash 启动的软件配合实现

现在我们知道了 DM368 NAND Flash 启动的原理，下面我们来看看软件是如何根据并配合硬件的要求实现启动的。在 DM368 IPNC 的软件包里面有一个工具的目录，里面有预先编译好的烧写 NAND 的 CCS 的可执行文件，UBL 的二进制文件以及相关源码。

2.1 UBL 描述符的实现

刚才在介绍 NAND Flash 启动原理的时候，我们提到了 RBL 需要到 NAND Flash 上面搜索特殊数字标志。这个特殊数字标志就是由烧写 NAND 的 CCS 的工程写到 Flash 上的。在 flash_utils_dm36x_1.0.0\flash_utils_dm36x\DM36x\CCS\NANDWriter\src\nandwriter.c 里面的 LOCAL_writeHeaderAndData()函数就是用来写描述符的。

```
// Setup header to be written
  headerPtr = (Uint32 *) gNandTx;
  headerPtr[0] = nandBoot->magicNum;          //Magic Number
  headerPtr[1] = nandBoot->entryPoint;        //Entry Point
  headerPtr[2] = nandBoot->numPage;           //Number of Pages
#if defined(IPNC_DM365) || defined(IPNC_DM368)
  headerPtr[3] = blockNum+3;                  //Starting Block Number
   headerPtr[4] = 0;                           //Starting Page Number - always start data in
page 1 (this header goes in page 0)
```

对比表 1，你可看到 headerPtr[3]的内容是用来存放 UBL 代码的起始块号。这里+3 的意思就是 UBL 是存放在 UBL 描述符所放块号后面的第三块里面。headerPtr[4] = 0 表示是从第 0 页开始存放。当然这个值用户是可以修改的。只要你烧写 UBL 代码的位置和描述符里面的起始块/页数一致就可以了。

在 IPNC 的代码里面 UBL 的描述符是会从 NAND Flash 的第 1 个块开始写，如果块是好的，就放在第 1 块的第 0 页。如果第 1 块是坏的，就会把 UBL 的描述符写入到下一个块的第 0 页。IPNC 的代码里面没有将 UBL 描述符可能有的块号从 1 到 24 块（这是 RBL 搜索的范围），它只是从第 1 块到第 3 块。如果 UBL 描述符放在第 1 块，那如果第 4 块是好的话，UBL 的代码就从第 4 块的第 0 页开始放。

```
#elif defined(IPNC_DM368)
// Defines which NAND blocks the RBL will search in for a UBL image
#define DEVICE_NAND_RBL_SEARCH_START_BLOCK     (1)
#define DEVICE_NAND_RBL_SEARCH_END_BLOCK       (3)
```

在 nandwriter.c 里面你还可以看到 UBL 的入口地址是固定的 0x100。

```
gNandBoot.entryPoint  = 0x0100;       // This fixed entry point will work with the UBLs
```

要了解为什么是 0x100，你就必须要看一下 UBL 的源码。在 UBL 源码的 UBL.cmd 文件里面，你可以看到下面的定义，将入口地址放在 boot 的地方，而 boot 的运行地址就是 0x100。

```
-e boot //指定入口地址为 boot
…
MEMORY
{
  …
  UBL_I_TEXT      (RX)   : origin = 0x00000100   length = 0x00004300
…
  UBL_F_TEXT      (R)    : origin = 0x020000E0   length = 0x00004300
  ...
}

SECTIONS
{
  …
  .text : load = UBL_F_TEXT, run = UBL_I_TEXT, LOAD_START(FLASHTEXTStart),
LOAD_SIZE(FLASHTEXTSize)
  {
    *(.boot)
    . = align(4);
    *(.text)
    . = align(4);
  }
  ….
}
```

在 UBL 的源码 boot.c 里面有强制把启动的最初代码放在了 boot 的 section 里面。

```
#if defined(__TMS470__)
  …
  #pragma CODE_SECTION(boot,".boot");
#endif
void boot(void)
{
…
}
```

这样从 cmd 的配置以及代码指定代码段，UBL 的程序就能确保是从 0x100 的地址开始运行。

2.2 U-Boot 启动实现

UBL 启动 U-Boot 的过程，借鉴了 RBL 启动 UBL 的原理。烧写描述符也是用同样的 LOCAL_writeHeaderAndData()函数。在 nandwriter.c 里面，我们把 U-Boot 的代码叫做应用代码（APP）。

```
// Defines which NAND blocks are valid for writing the APP data
#define DEVICE_NAND_UBL_SEARCH_START_BLOCK     (8)
#define DEVICE_NAND_UBL_SEARCH_END_BLOCK       (10)
```

下面是 IPNC 启动后串口最初的打印。

```
Valid magicnum, 0xA1ACED66, found in block 0x00000008.
   DONE
Jumping to entry point at 0x81080000.
```

我们可以看到 UBL 是指第 8 块的地方找到了 U-Boot 的描述符，这个和 DEVICE_NAND_UBL_SEARCH_START_BLOCK 的定义是一致的。

2.3 U-Boot 更新 UBL 和 U-Boot 的原理

IPNC 代码支持在 U-Boot 里面更新 UBL 或者 U-Boot 自己。下面是烧写 ubl 和 U-Boot 在 U-Boot 下的命令。

```
烧写 ubl:
nand write 0x80700000 0x080000 0x08000
烧写 U-Boot:
nand write 0x80700000 0x160000 0x28000
```

要了解为什么 NAND Flash 的烧写地址是 0x80000 和 0x160000，这还是需要了解 nandwriter.c 里面的烧写流程。从前面的内容我们可以得知，nandwriter.c 烧写 UBL 是从 1+3=4 块开始的，而烧写 U-Boot 是从 8+3=11 块。在 IPNC 上使用的 NAND Flash 是 2K 一个页，每个块 128KB。所以 UBL 烧写的地址是 128KBx4=0x80000，而烧写 U-Boot 的地址是 128Kx11=0x160000。

2.4 NAND Flash 没有坏块的情况

所以如果在没有 NAND Flash 坏块的情况下，nandwriter.c 会把 UBL 的描述符烧写在第 1 块第 0 页上，把 UBL 的代码烧写在第 4 块第 0 页上，把 U-Boot（APP）的描述符烧写在第 8 块第 0 页上，把 U-Boot 的代码烧写在第 11 块第 0 页上。这样芯片在上电确认是 NAND Flash 启动后，RBL 在执行的时候就会找到 UBL 相应的描述符，把 UBL 加载的 ARM 内存里运行。而 UBL 又找到了 U-Boot 的描述符，把 U-Boot 加载到 DDR 上运行。最后 U-Boot 加载 uImage 并启动了 Linux，完成了从上电到 Linux 启动的整个过程。

3 结束语

每个芯片一般都有多种启动方式，各个芯片的启动方式都有所不同，但又有类似的地方。上面的介绍也可以作为学习其他芯片其他启动方式的一个参考。

最后感谢李斌在本文整理过程中的帮助。

参考文档

1. TMS320DM36x Digital Media System-on-Chip (DMSoC) ARM Subsystem User's Guide Literature Number: SPRUFG5A
2. TMS320DM368 datasheet Literature Number: SPRS668C

重要声明

德州仪器(TI) 及其下属子公司有权在不事先通知的情况下, 随时对所提供的产品和服务进行更正、修改、增强、改进或其它更改，并有权随时中止提供任何产品和服务。客户在下订单前应获取最新的相关信息，并验证这些信息是否完整且是最新的。所有产品的销售都遵循在订单确认时所提供的TI 销售条款与条件。

TI 保证其所销售的硬件产品的性能符合TI 标准保修的适用规范。仅在TI 保证的范围内，且TI 认为有必要时才会使用测试或其它质量控制技术。除非政府做出了硬性规定，否则没有必要对每种产品的所有参数进行测试。

TI 对应用帮助或客户产品设计不承担任何义务。客户应对其使用TI 组件的产品和应用自行负责。为尽量减小与客户产品和应用相关的风险，客户应提供充分的设计与操作安全措施。

TI 不对任何TI 专利权、版权、屏蔽作品权或其它与使用了TI 产品或服务的组合设备、机器、流程相关的TI 知识产权中授予的直接或隐含权限作出任何保证或解释。TI 所发布的与第三方产品或服务有关的信息，不能构成从TI 获得使用这些产品或服务的许可、授权、或认可。使用此类信息可能需要获得第三方的专利权或其它知识产权方面的许可，或是TI 的专利权或其它知识产权方面的许可。

对于TI 的产品手册或数据表，仅在没有对内容进行任何篡改且带有相关授权、条件、限制和声明的情况下才允许进行复制。在复制信息的过程中对内容的篡改属于非法的、欺诈性商业行为。TI 对此类篡改过的文件不承担任何责任。

在转售TI 产品或服务时，如果存在对产品或服务参数的虚假陈述，则会失去相关TI 产品或服务的明示或暗示授权，且这是非法的、欺诈性商业行为。TI 对此类虚假陈述不承担任何责任。

TI 产品未获得用于关键的安全应用中的授权，例如生命支持应用（在该类应用中一旦TI 产品故障将预计造成重大的人员伤亡），除非各方官员已经达成了专门管控此类使用的协议。购买者的购买行为即表示，他们具备有关其应用安全以及规章衍生所需的所有专业技术和知识，并且认可和同意，尽管任何应用相关信息或支持仍可能由TI 提供，但他们将独力负责满足在关键安全应用中使用其产 品及TI 产品所需的所有法律、法规和安全相关要求。此外，购买者必须全额赔偿因在此类关键安全应用中使用TI 产品而对TI 及其 代表造成的损失。

TI 产品并非设计或专门用于军事/航空应用，以及环境方面的产品，除非TI 特别注明该产品属于“军用”或“增强型塑料”产品。只 有TI 指定的军用产品才满足军用规格。购买者认可并同意，对TI 未指定军用的产品进行军事方面的应用，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

TI 产品并非设计或专门用于汽车应用以及环境方面的产品，除非TI 特别注明该产品符合ISO/TS 16949 要求。购买者认可并同意，如果他们在汽车应用中使用任何未被指定的产品，TI 对未能满足应用所需要求不承担任何责任。

可访问以下URL 地址以获取有关其它TI 产品和应用解决方案的信息：

产品		应用	
数字音频	www.ti.com.cn/audio	通信与电信	www.ti.com.cn/telecom
放大器和线性器件	http://www.ti.com.cn/amplifiers	计算机及周边	www.ti.com.cn/computer
数据转换器	http://www.ti.com.cn/dataconverters	消费电子	www.ti.com/consumer-apps
DLP® 产品	www.dlp.com	能源	www.ti.com/energy
DSP - 数字信号处理器	http://www.ti.com.cn/dsp	工业应用	www.ti.com.cn/industrial
时钟和计时器	http://www.ti.com.cn/clockandtimers	医疗电子	www.ti.com.cn/medical
接口	http://www.ti.com.cn/interface	安防应用	www.ti.com.cn/security
逻辑	http://www.ti.com.cn/logic	汽车电子	www.ti.com.cn/automotive
电源管理	http:///www.ti.com.cn/power	视频和影像	www.ti.com.cn/video
微控制器 (MCU)	http://www.ti.com.cn/microcontrollers	无线通信	www.ti.com.cn/wireless
RFID 系统	http://www.ti.com.cn/rfidsys		
RF/IF 和 ZigBee® 解决方案	www.ti.com.cn/radiofre		
	TI E2E 工程师社区	http://e2e.ti.com/cn/	

邮寄地址： 上海市浦东新区世纪大道 1568 号，中建大厦 32 楼 邮政编码： 200122
Copyright © 2011 德州仪器 半导体技术（上海）有限公司