Bootloader 開發

■ 摘要

啟動程式（boot loader）位於電腦或其他計算機應用上，是指啟動作業系統的程式，在嵌入式系統中，功用為初始化硬體、最終目標啟動 kernel(在此專案為 linux kernel)。之前在開發版上使用過 u-boot 當作嵌入式 linux 的啟動程式，並且花了一點時間去閱讀源碼，理解其架構，這次就利用對其的理解，試著做一個小型的 bootloader，功能大致有初始化硬體 ex: 看門狗、SDRAM、Nand flash 等，在最後透 linux 開發源碼中 setup.h 文件裡面的 tag 結構傳遞內核參數啟動內核。

■ 目錄

1. 嵌入式 linux 結構與分布

一般情況下嵌入式 Linux 系統中的軟件主要分為以下幾部分：

1. **引導程式**：此部分有兩個階段，第一個階段為固化 ROM 是晶片廠家在出產的時候燒入到 ROM 裡面的啟動程式，例如有些開發版可以選擇 nand flash 啟動或者是 nor flash 啟動。
2. **Linux 內核和驅動程式**。
3. **文件系統**:

包括根文件系統和建立在 Flash 設備上的文件系統（EXT4，UBI，CRAMFS 等等）。提供管理系統的各種配置文件以及系統執行用戶應用程序的良好運行環境及載體。

4. **應用程序**。用戶自定義的應用程序，存放於文件系統之中。

2. 規劃 bootloader memory

Bootloader 裝載的開發版為 S32440 Soc，具有 nand flash 及 nor flash 可以存放程式碼，容量大小分別為 64M(nand)、2M(Nor)。我選擇使用較大容量的 nandflash 放置我們的 bootloader 程式，並且驅動 nandflash 將存放在 nand 底下的內核(uImage)讀到 SDRAM 執行

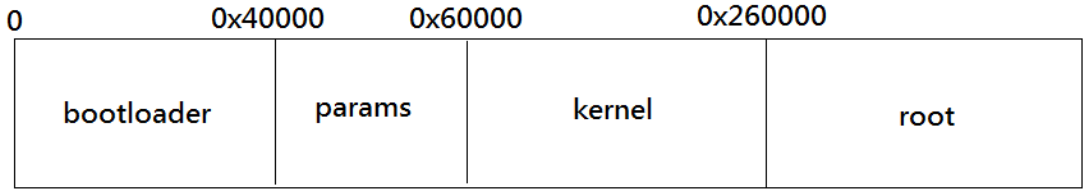

圖 1 Nand flash memory 大致規劃

我所使用的開發版在 nand flash 啟動時 廠家固化 ROM 裡面的 bootloader 會將 nand flash 前面 4K 大小的程式載入到內部 RAM 運行啟動，如果說程式碼大於 4K 就有必要驅動 SDRAM(外部的 RAM)將程式碼從 nand flash 讀到 SDRAM 運行。

由上面的結果可以得知，我們的內核大小為 2M 是無法在內部 RAM 上自動運行的需要我們驅動 nand flash 將 nand flash 0x60000 地址存放的內核讀到 SDRAM 上運行我們在此專案內核鏈接在 SDRAM 的地址為 0x30007FC0。詳細規劃如圖 2 所示

3. 建構鏈接腳本

有了初步的記憶體空間規畫，便可開始撰寫鏈接腳本，以下為 CSDN 對鏈接腳本的敘述。

GNU-ld 鏈接腳本淺析

每一個鏈接過程都由鏈接腳本(linker script, 一般以 lds 作為文件的後綴名)控制. 鏈接腳本主要用於規定如何把輸入文件內的 section 放入輸出文件內, 並控制輸出文件內各部分在程序地址空間內的佈局,連接器有個默認的內置連接腳本，-T 選項用以指定自己的鏈接腳本, 它將代替默認的連接腳本。你也可以使用<暗含的連接腳本>以增加自定義的鏈接命令。

由以上的敘述加上之前規畫的記憶體地址我們 bootloader 決定放在 SDRAM 叫頂部的位置，避免與內核有衝突

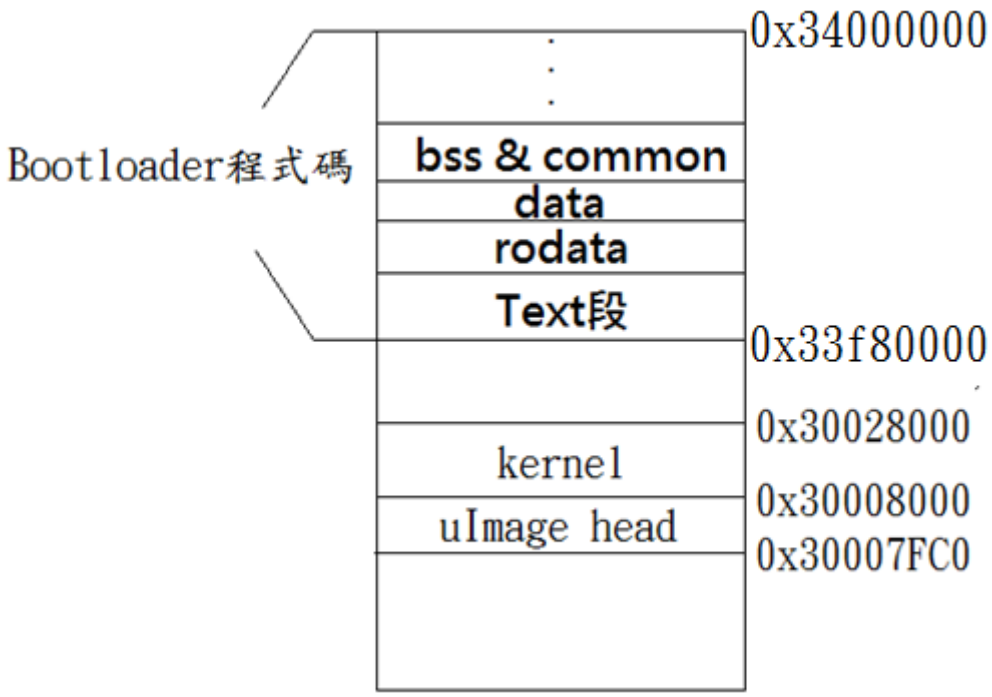

圖 2 SDRAM memory 分布圖

4. Bootloadr 第一階段

在多數的單晶片程式中，一開始的程式碼片段多半都是驅動一些必要的硬體，且是用組合語言完成，我們撰寫的 bootloader 也不意外，在啟動時必須先使用組合語言驅動需要使用的硬體設備，在本階段中分別是關閉看門狗、初始化時鐘頻率(因為需要用到 UART 做調適)、初始化 SDRAM、最後要設置好 SP 也就是堆疊指針才可以跳轉到 C 語言程式碼完成更複雜的硬體初始化。

在 C 語言程式碼中，我實現的 nand flash 的操作並且將程式碼從 nand flash 中將 bootloader 程式碼讀到 bootloader 所在的鏈接腳本(0x33f80000)，使程式可以在 SDRAM 用更大的 RAM 順利執行。

5. Bootloader 第二階段

在第一階段的 bootloadr 初始化一些必要的硬體設備，接下來就只剩下啟動 linux kernel 了，在這個階段，我將參考 u-boot 跳轉 linux kernel 的程式碼結構撰寫。以下為撰寫的程式碼片段

```
#define nand_kernel_addr 0x60000
#define kernel_addr 0x30007FC0
#define kernel_size 0x200000

#define MACH_TYPE_S3C2440   362
int main(void)
{
    void (*theKernel)(int, int, unsigned int);
    nand_read(nand_kernel_addr + 64,(unsigned char*)(kernel_addr + 64),kernel_size);


    putstr("set parameter\r\n");
    setup_start_tag();
    setup_memory_tags();
    setup_commandline_tag("noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0");
    setup_end_tag();

    putstr("Start kernel.....\r\n");
    theKernel = (void (*)(int, int, unsigned int))(kernel_addr + 64);
    theKernel(0,MACH_TYPE_S3C2440,params_addr);

    putstr("error\r\n");
    while(1);
    return 0;
}
```

圖 3 Bootloader 第二階段程式碼

從 u-boot 中可以得知，再啟動 linux 內核時需要將傳遞的參數存放在某個地址，當內核啟動後程式將會到該地址取出參數並且按照參數的設定順利啟動內核。並且得知內核參數會用一種特定的結構體保存此結構圖如下所示

```
struct tag {
        struct tag_header hdr;
        union {
                struct tag_core               core;
                struct tag_mem32        mem;
                struct tag_videotext        videotext;
                struct tag_ramdisk        ramdisk;
                struct tag_initrd        initrd;
                struct tag_serialnr        serialnr;
                struct tag_revision        revision;
                struct tag_videolfb        videolfb;
                struct tag_cmdline        cmdline;
                /*
                 * Acorn specific
                 */
                struct tag_acorn        acorn;
                /*
                 * DC21285 specific
                 */
                struct tag_memclk        memclk;
        } u;
};
```

圖 4 Tag 結構體

透過 tag 標籤結構體，我們可以存放參數在特定的地址，內核可以從這個地址讀取參數啟動，我傳遞的內核參數分別為起始標籤、記憶體標籤、命令標籤、結束標籤，分別對應 setup_start_tag、setup_memory_tags、setup_commandline_tag、setup_end_tag 函式。

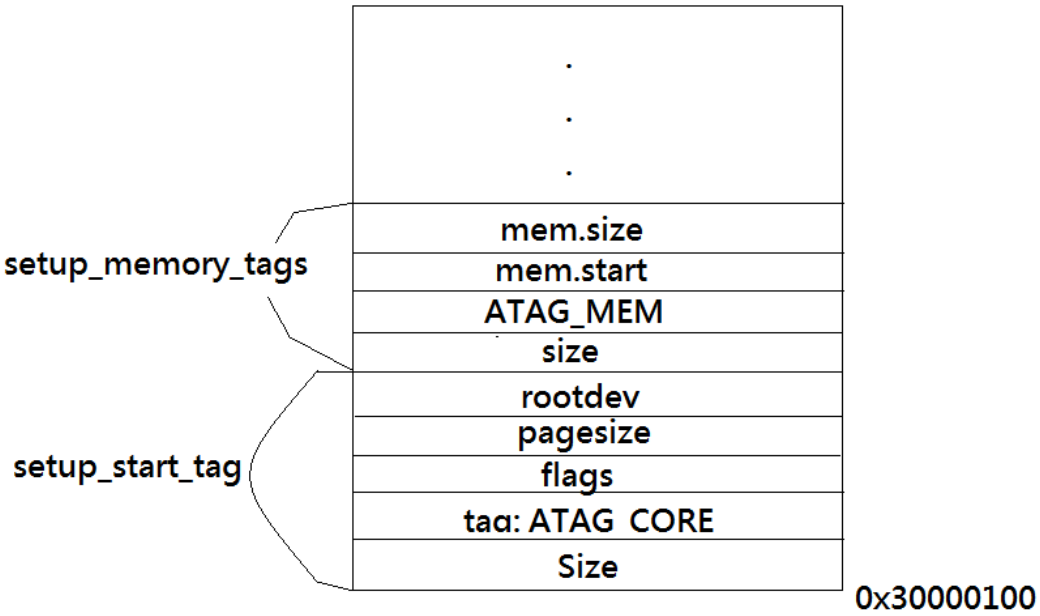

圖 5 SDRAM tag memory 分佈

最後透過以下函式啟動內核

```
void (*theKernel)(int, int, unsigned int);
theKernel = (void (*)(int, int, unsigned int))(kernel_addr + 64);
theKernel(0,MACH_TYPE_S3C2440,params_addr);
```

至於為什麼要跳轉到地址 0x30007FC0 偏移 64bytes 的地址運行內核呢?
原來是內核編譯出來的映象檔案室 uImage 格式

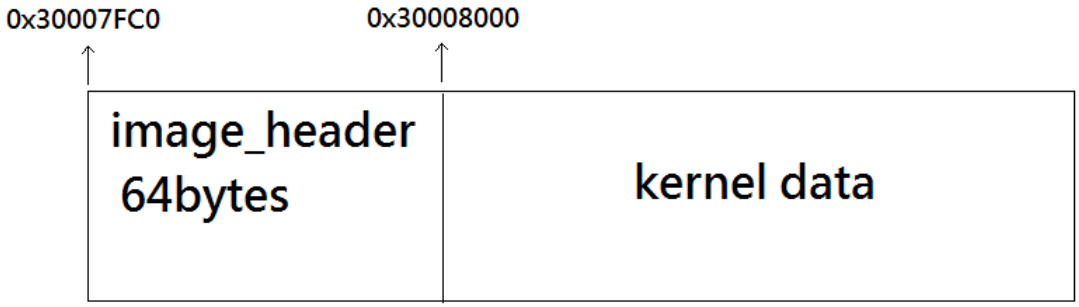

圖 6 uImage 格式

在 uImage 格式中會在內核資料前面 64bytes 空間填入一些資訊，例如內核運行的地址等等，但是因為我完成的 bootloader 為一個簡單的版本，並沒有去分析開頭 64bytes uImage header 的資訊，因此跳轉到主要的內核地址運行即可。

參考文獻

1. GNU-ld 鏈接腳本淺析
2. u-boot 1.1.6
3. s3c2440 datasheet