

基于 P2020 的 SPI FLASH 模式的研究与实现

王 军,胡 明

(电子科技大学 通信与信息工程学院,四川 成都 611731)

摘 要:PowerPC P2020 是 Freescale 公司一款性能优良的处理器,在电信、军事、网络等领域都有着广泛的应用。SPI FLASH 作为一种重要的启动方式,是用户在进行硬件设计时经常采用的方法之一。它不仅丰富用户对于启动方式的选择,同时也可以作为备份启动模式,提高系统的稳定性。鉴于 SPI FLASH 启动方式的重要性,文中以 P2020 为硬件平台,对 SPI FLASH 启动方式的实现进行了研究。首先,描述了 P2020 处理器上电之后 SPI FLASH 两个阶段的启动过程;其次,介绍了 SPI FLASH 启动镜像的制作过程;再次,针对实际操作 SPI FLASH 中遇到的一些问题,文中进行了有效地解决;最后,在完成 uboot 及 linux 的移植后,最终成功实现了 SPI FLASH 启动方式。系统上电之后,可成功启动 uboot 并加载 linux,且可以稳定运行。

关键词:PowerPC;P2020 处理器;SPI 闪存;启动方式;uboot

中图分类号:TP316

文献标识码:A

文章编号:1673-629X(2013)05-0154-05

doi:10.3969/j.issn.1673-629X.2013.05.040

Research and Implementation of SPI FLASH Mode Based on P2020

WANG Jun, HU Ming

(College of Communication & Information Engineering, UESTC, Chengdu 611731, China)

Abstract:PowerPC P2020 is a high-performance processor of Freescale Semiconductor, which has a wide range of applications in areas such as telecommunications, military, network. As an important boot media, SPI FLASH is one of the most used methods when users design their hardware. It can enrich the user's choice for the boot mode. At the same time, as a backup boot mode, it can also improve the stability of the system. In view of the importance of SPI FLASH boot mode, based on the P2020 platform, conduct a research on the implementation of booting from SPI FLASH. Firstly, describe the two boot stages of SPI FLASH after P2020 power on. Secondly, introduce the process of building the SPI FLASH image used for booting. Thirdly, for some of the problems encountered in the actual operation on SPI FLASH, give effective solutions. Finally, after the transplantation of uboot and linux, implement the booting from SPI FLASH. After power on, the system can boot the uboot and load linux successfully, and run within a stable condition.

Key words:PowerPC;P2020 processor;SPI FLASH;boot mode;uboot

0 引 言

随着高速数据传输系统应用的日益广泛,PowerPC 系列处理器以其卓越的通信能力与高度的稳定性,在工业控制高速通信及数据存储领域具有重要的应用^[1]。

PowerPC 架构的特点是可伸缩性好,方便灵活。PowerPC 处理器品种很多,既有通用的处理器,又有嵌

入式控制器和内核,从高端的工作站、服务器到桌面计算机系统,从消费类电子产品到大型通信设备,应用范围非常广泛^[2]。

P2020 处理器是 Freescale 公司 PowerPC 架构下 QorlQ P2 平台的一款核心产品。它采用双核高性能 Power Architecture™ E500 v2 核心,具有 1.2GHz 的时钟频率,同时支持 DDR2 和 DDR3,提供三个 eTSEC 接口,3 个 PCIe 控制器。可广泛应用于电信、网络、军事等各领域^[3]。

引导加载程序(Bootloader)是在操作系统内核运行之前执行的一段小程序,通过这段程序初始化硬件设备、建立内存空间的映射表和传递给操作系统一些基本的配置参数,建立起操作系统运行的环境,为调用操作系统内核做好准备。uboot 是当前嵌入式中较前

收稿日期:2012-08-13;修回日期:2012-11-25

基金项目:中央高校基本科研业务费专项资金资助项目(ZYGX2009J008)

作者简介:王 军(1986-),男,河南南阳人,硕士研究生,研究方向为通信与信息系统、嵌入式系统;胡 明,副教授,硕士生导师,研究方向为通信网与宽带通信技术、光分组传输网、DWDM。

沿的功能完善的 Bootloader^[4]。它是遵循 GPL 条款的 Bootloader 开放源码项目,支持 PowerPC、MIPS、x86 和 ARM 等系列的处理器,能够加载 Linux、VxWorks 和 QNX 等多种目标操作系统^[5]。uboot 常被用作 PowerPC 处理器的引导加载程序^[6],文中采用 uboot 作为 Bootloader。

通常而言,uboot 可以从不同的存储介质上启动,现在经常采用的有 3 种模式,NOR FLASH 启动,NAND FLASH 启动,SPI FLASH 启动。

NOR FLASH 启动模式,可以进行字节访问,且可以片内执行,是启动设备的最佳选择。但随着嵌入式设备的功能越来越复杂,对启动介质的容量要求也越来越大,NOR FLASH 容量有限的缺点逐渐显现。NAND FLASH 和 SPI FLASH 启动模式越来越受欢迎。

从硬件角度来讲,NOR FLASH 需要 8 位或者 16 位的数据总线及单独的地址总线。NAND FLASH 也需要 8 位或者 16 位的总线,数据和地址线复用。而 SPI 启动方式只需要简单的 4 位串行总线。相对而言,采用 SPI FLASH 对硬件设计的要求更低,对硬件资源的消耗也较少。

同时,SPI FLASH 经常做为备份启动方式,在主启动方式失效时,对系统提供稳定的支持。因此,SPI FLASH 启动方式在实际中也得到了广泛的应用。

文中首先介绍了 P2020 上电之后的启动过程,以此为理论基础,进而介绍了 SPI FLASH 启动模式的最终实现。

1 P2020 上电启动过程分析

P2020 采用 E500 core。E500 core 在上电复位时,系统会到地址 FFFF_FFFC 处读取第一条指令,该指令一般是一个跳转指令,跳转到 arch\powerpc\cpu\mpc85xx\start.S 中 bootpg 部分的汇编代码头部去执行^[7,8]。

从功能层面上分析,uboot 的启动过程可以分为两个阶段:

(1) 第一阶段。

在 SPI FLASH 中执行。主要完成硬件设备(主要为 CPU、RAM 和串口等)的初始化,加载 uboot 第二阶段代码到 RAM 空间中(代码搬运),分配堆栈空间,清零 BSS 数据段,构建 C 语言运行环境,跳转到第二阶段入口函数等。

(2) 第二阶段。

在 RAM 中执行。第二阶段可以执行功能更为复杂的初始化工作,同时执行速度也更快。初始化本阶段使用的硬件设备,主要是进行 CPU 及外设的进一步初始化,如读取网卡的配置信息,初始化网口,PCI、

PCIE、串口、网口、Flash 等设备的初始化等等。在设备初始化完成后,便会进入 uboot 中的 main_loop 循环中,等待用户输入,执行相应命令,或者为内核设置启动参数,自动启动系统。

从代码层面上分析,E500 核的 uboot 的启动是从 arch\powerpc\cpu\mpc85xx\start.S 中的 _start_e500 标号处开始。经历若干文件的汇编和 C 语言函数后,在 arch\powerpc\lib\board.c 中的 board_init_r 函数进入 main_loop 循环处结束,如图 1 所示。

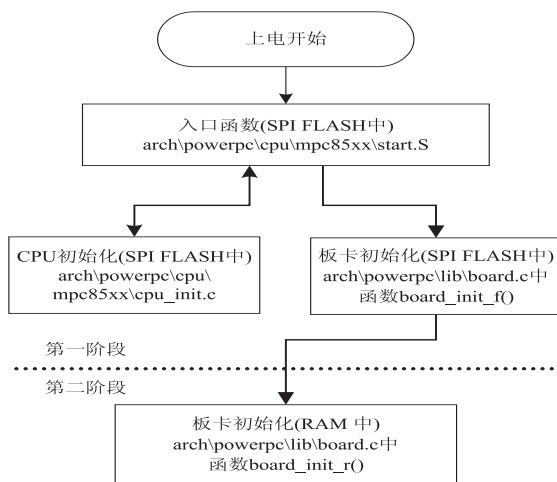


图 1 P2020 上电启动过程

如图 1 所示,虚线为两个阶段的分界线。上电启动之后,系统首先从 start.S 中跳转到 cpu_init.c 中进行 CPU 的初始化,完成后跳转回 start.S 中。然后,从 start.S 中跳转到 board.c 中,调用函数 board_init_f 进行板卡的第一次初始化,启动的第一阶段完成。第二阶段主要调用函数 board_init_r 进行板卡的第二次初始化,完成整个启动过程。

下面对两个阶段的启动过程进行说明。

1.1 启动的第一阶段

在第一阶段里面,系统首先从 start.S 中跳转到 cpu_init.c 文件中,进行 CPU 的初始化。arch\powerpc\cpu\mpc85xx\cpu_init.c 调用函数 cpu_init_f。此函数是系统执行的第一个 C 语言的函数,进行 CPU 底层的初始化^[9],主要是做一些 CPU 寄存器的初始化,其中最重要的部分是初始化 Local Access Windows 的值,Local Bus 上的片选 BR、OR 的值和配置 MMU 的 LTB1、LTB0,初始化 DMA 等。

CPU 初始化完成后,重新回到 start.S 中,跳转到 board.c 文件中,调用函数 board_init_f 进行主板的第一次初始化,仍在 SPI FLASH 中进行。该函数主要为板子上的硬件做初始化工作,通过依次调用函数指针数组 init_sequence[] 中的函数完成。

init_sequence 数组里面包含了很多板级硬件的初始化函数,如: init_baudrate 波特率初始化设置, serial_

init 串口初始化, checkcpu 打印 CPU 信息, checkboard 显示板卡信息, init_func_ram RAM 的初始化等等。

执行代码如下所示:

```
for (init_fnc_ptr = init_sequence; * init_fnc_ptr; ++init_fnc_ptr) {  
    if ((* init_fnc_ptr)() != 0) {  
        hang();  
    }  
}
```

此处的 for 循环结束后, 已完成主要硬件设备的初始化, 尤其是内存 RAM 的初始化工作已完成, C 语言的运行环境已经准备完毕, 可以向 RAM 中复制代码, 于是调用函数 relocate_code。

relocate_code 函数主要功能有两个:

一是从 Flash 中拷贝 u-boot 的代码到 RAM 当中;
二是记录现在执行代码的偏移, 跳转到 RAM 中相应的位置去执行。

函数 relocate_code 调用完成后, 启动的第一阶段结束。

1.2 启动的第二阶段

启动的第二阶段, 在内存中进行, 主要调用函数 board_init_r 进行板卡的第二次初始化。

函数 board_init_r 进而会调用一系列的函数来完成硬件的初始化工作, 如: 调用函数 serial_initialize 完成串口的初始化; 调用函数 spi_init_r 完成 SPI 控制器的初始化; 调用函数 mmc_initialize 完成 MMC 的初始化; 调用函数 pci_init 完成 PCI 设备的初始化; 调用函数 interrupt_init 完成中断的初始化等等。

系统初始化完成之后, 进入 main_loop 命令行状态, 如下:

```
for (;) {  
    WATCHDOG_RESET();  
    main_loop();  
}
```

main_loop() 等待用户输入命令并执行, 或者按照内核启动参数自动加载操作系统。至此, P2020 的上电启动过程完毕。

2 SPI FLASH 启动镜像的制作

基于前面对 P2020 上电启动过程的分析, 下面介绍在 P2020 硬件平台上, SPI FLASH 启动镜像的制作。文中所使用的处理器型号为 P2020NSE2MFC, 时钟频率为 1.2GHz, 使用 2GB DDR3。在进行移植时, 所使用 uboot 版本为: u-boot-2010.09; 所使用 linux 内核版本为: linux-2.6.35.9。

(1) 生成 u-boot. bin 二进制文件。

uboot 的移植工作, 主要是参考 Freescale 公司官方

所提供的参考设计板及相关数据手册, 根据硬件平台实际的设计, 进行有针对性地修改完善而成。此处不再详述其过程。

在 uboot 已按照要求修改完毕之后, 现在就要进行编译, 生成 u-boot. bin 二进制文件, 方法如下:

首先配置板卡的编译环境:

make P2020RDB_SPIFLASH_config

再进行编译:

make -j 4

即生成 u-boot. bin 文件, 大小为 512KB。

(2) 配置文件 config_sram. dat 介绍。

现在已经有了启动的二进制镜像文件 u-boot. bin, 对于 SPI 启动, 还需要有配置文件, 对启动时的相关参数进行设定。

配置文件 config_sram. dat 格式如下所示:

40:424f4f54

44:00000000

48:00080000

每一个配置字, 前面为偏移地址, 后面为对应的配置信息。这些语句指定启动的关键参数, 如 uboot 镜像的大小, 开始读取用户数据的起始地址等。下面对其中一些参数进行简单介绍:

0x40: booting signature 值, 一般为 424f4f54, 是 BOOT 的 ASCII 值表示;

0x48: uboot. bin 镜像文件的大小, 一般为 512KB, 所以值为 80000;

0x50: 指定 uboot 启动代码的源地址, 一般为 1000, 即表示 SPI FLASH 中地址 1000 后面为 uboot 的启动代码;

0x58: 指定 uboot 代码拷贝到内存中的位置, 32 位地址, 如 f8f80000;

0x68: 指定其后面配置语句的对数^[10]。

(3) SPI FLASH 启动镜像 spiimage 的生成。

欲生成 spiimage, 需使用 Boot_format 工具。Boot_format 是 Freescale 公司所提供的 LTIB 中众多工具的一种, 专门用来生成启动时所用的镜像文件。

使用此工具制作 spiimage, 参数设置如下^[10]:

./boot_format config_sram. dat u-boot. bin -spi spiimage. bin

注: config_sram. dat 是使用的配置文件, uboot. bin 是生成的 uboot 二进制文件, -spi 是参数, spiimage. bin 是最终生成的文件。

至此, SPI FLASH 启动 uboot 的镜像文件已经生成。

(4) 烧入 spiimage。

在 NAND FLASH 启动的 uboot 下, 将 spiimage 启

动镜像文件烧入 SPI FLASH 中,命令示例如下:

```
setenv serverip 192.168.1.100;  
tftp 100000 spiimage.bin;  
sf probe 0 40000000;  
sf erase 0 80000;  
sf write 100000 0 80000;
```

3SPI FLASH 关键性能的优化

SPI FLASH 启动镜像烧入之后,设置板卡的跳线方式,选择 SPI FLASH 启动模式,可以启动 uboot。但在 uboot 下实际操作 SPI FLASH 的过程中,发现存在一些需要优化改进的地方,如不能读取大数据块、读写操作较为缓慢等,文中逐一进行了解决。

2.1 不能读取大块数据的问题

SPI FLASH 成功启动 uboot 之后,在测试 SPI FLASH 功能时,发现在读写小块数据(小于 32KB)时,可正常写入,且可以正常读出。但是在读取大块数据时,出现错误。

反复排查之后,发现 uboot 中对一次读取的数据长度有所限制。所使用的 SPI FLASH 来自 spanion 公司,型号为 S25FL128P,容量为 16MB,扇区大小为 256KB,页大小为 256B^[11]。其进行数据读取时按帧进行,每帧数据最大限制为 32KB。通过定位,发现此部分功能是在 common/cmd_sf.c 文件中的函数 static int do_spi_flash_read_write() 中予以实现的。此函数实现对 SPI FLASH 的循环读取。

经分析代码之后,做出以下修改:

(1)在文件 cmd_sf.c 中增加宏定义:

```
#define FRAME_SIZE15
```

(2)在此函数变量声明部分增加:

```
unsigned int framecnt//需读取的总帧数
```

```
unsigned frame_size//每帧的大小
```

```
unsigned i//循环变量
```

(3)在 spi_flash_read() 函数执行读操作前增加:

```
framecnt = (len >> FRAME_SIZE) + 1;
```

计算需读取的帧数;

(4)原有的 ret = spi_flash_read(flash, offset, len, buf) 替换为:

```
ret = spi_flash_read(flash, (unsigned int)offset + i  
* frame_size, (framecnt - i - 1 == 0) ? len - i * frame_size :  
frame_size, (unsigned int)buf + i * frame_size);
```

即重新计算 SPI FLASH 执行读操作时的偏移地址、长度等。

修改之后,重新编译 uboot,烧入之后,SPI FLASH 在读取大于 32KB 的数据块时,功能正常。

2.2 读写速度缓慢的问题

在对 SPI FLASH 执行读写操作时,耗时很大,远远

大于芯片数据手册中所给出的延时。

分析后发现:在执行读写操作时,是使用 driver/spi/fsl_espi.c 文件中的 espi_xfer() 函数,其中,espi 传送一次数据就要延迟 80 微秒。

于是修改此函数,去掉延迟语句 udelay(80),并在其前加入 while 循环语句:

```
while(!((espi->event >> 26) || (espi->event &  
(1 << 14))));
```

注:此处所加入的 while 语句主要是参考了 P2020 数据手册中 eSPI 事件寄存器部分后,所做出的修改。通过判断 eSPI 事件寄存器标志位的值,来确定收发状况,从而替代简单的延时语句,达到减小延迟的目的。

经过测试,修改之后 SPI FLASH 在执行读写操作时,速度已经符合数据手册中所给出的参考值。

2.3 擦除操作不精确的问题

在对 SPI FLASH 进行擦除操作时,不能按指定的长度进行。表现在,操作 Flash 的高位地址时擦除失败。

经分析,在 uboot 目录/drivers/mtd/spi 下 span-sion.c 文件中,定义了对 Flash 进行的基本操作,由于其默认擦除的扇区大小为 64K,而所使用的 Flash 为 256KB。所以需做以下修改:

在函数 spansion_erase() 中,将

```
cmd[1] = (offset/sector_size) + actual;
```

修改为:

```
cmd[1] = (offset/(64 * 1024)) + actual * sector_  
size/(64 * 1024);
```

修改主要着眼于修正所使用的扇区大小,使执行擦除操作时更为精准。重新编译之后,烧入 SPI FLASH 中测试,可以实现正常擦除。

3 SPI FLASH 启动方式的实现

在解决了实际操作 SPI FLASH 中遇到的一些问题之后,SPI FLASH 的各项功能归于正常,SPI FLASH 启动镜像已臻完善,重新烧写入 SPI FLASH 中,启动 uboot 后,打印信息如下所示:

```
U-Boot 2010.09 (Jun 28 2012 - 20:18:41)  
CPU0:P2020E, Version: 1.0, (0x80ea0010)  
Core:E500, Version: 4.0, (0x80211040)  
Clock Configuration:  
CPU0:1200 MHz, CPU1:1200 MHz,  
CCB:600 MHz,  
DDR:400 MHz (800 MT/s data rate) (Asynchronous), LBC:  
37.500 MHz  
L1:D-cache 32 KB enabled  
I-cache 32 KB enabled  
Board: P2020RDB Rev
```



```
I2C: ready
SPI: ready
DRAM: Using DDR SPD. . .
DDR: 2 GiB (DDR3, 64-bit, CL=6, ECC off)
FLASH: I2:512 KB enabled
NAND:1024 MiB
MMC:FSL_ESDHC: 0
SF: Detected S25FL128P_256K with page size 256, total 16
MiB
```

由上述打印信息可知,板卡各项硬件设备功能正常,运行稳定。

成功启动 uboot 之后,再烧入内核文件 uImage,设置树文件 p2020rdb. dtb,及文件系统镜像 rootfs. ext2. gz. uboot。

```
设置系统启动环境变量如下[12]:
setenv hwconfig "usb1;dr_mode=host;usb2;dr_mode=host;
esdhc"
setenv bootargs root=/dev/ram rw console = ttyS1, $ baudrate
$ othbootargs;
setenv bootcmd "sf probe 0 40000000;sf read 2000000 80000
80000;sf read 3000000 100000 400000;sf read 4000000 900000
600000;bootm 3000000 4000000 2000000";
boot;
重启系统。系统在启动 uboot 后,自动加载环境
变量,启动内核,成功进入操作系统,打印信息如下:
Welcome to the LTIB Embedded Linux Environment
P2020RDB login: PHY: 0;01 - Link is Up - 1000/Full
[root@P2020RDB /root]# uname -r
2.6.35.9
[root@P2020RDB /root]#
至此,SPI FLASH 启动方式顺利实现。
```

4 结束语

文中在 P2020 硬件平台之上,对 SPI FLASH 启动模式的原理、实现与优化进行了详尽介绍。SPI FLASH 启动方式有着独特的优势,相信在嵌入式领域

必将发挥更大的作用。采用 SPI FLASH 启动 uboot 时,还存在着诸如不能保存环境变量等一些问 题,这也是今后努力与改进的方向。

参考文献:

[1] 白 潇,徐智勇,张 耀. 基于 PowerPC 架构的嵌入式 Linux 系统开发和 RAID 技术[J]. 仪器仪表用户,2012 (2):49-51.

[2] 袁堂夫. 嵌入式 PowerPC 处理器浅[C]//2003 国际有线电视技术研讨会论文集. 杭州:出版者不详,2003.

[3] Freescale Semiconductor Corp. QorIQ™ P2020 Communications Processor Product Brief[M]. America: Freescale Semiconductor,2009.

[4] 陈柏喜,曾桂根. 基于 omap5912 的 uboot 平台构建[J]. 中国新通信,2009(3):53-55.

[5] 郭建磊,杨厚俊. 基于 S3C44B0X 的 U-Boot 及 μClinux 的移植分析[J]. 计算机技术与发展,2009,19(4):13-15.

[6] 刘邦运,别红霞. PowerPC 平台的 U-Boot 启动分析和移植[J]. 微计算机信息,2010(23):172-174.

[7] Freescale Semiconductor Corp. QorIQ™ P2020 Integrated Processor Reference Manual[M]. America: Freescale Semiconductor,2009.

[8] Freescale Semiconductor Corp. PowerPC™ e500 Core Family Reference Manual[M]. America: Freescale Semiconductor, 2005.

[9] 王长清,林桂竹. 基于 PowerPC 双核处理器嵌入式系统 U-Boot 移植[J]. 河南师范大学学报(自然科学版),2010 (1):70-74.

[10] Freescale Semiconductor Corp. Booting from On-chip ROM (eSDHC or eSPI)[M]. America: Freescale Semiconductor, 2010.

[11] Spansion Inc. S25FL128P. pdf[M]. America: Spansion Inc., 2007.

[12] Freescale Semiconductor Corp. BSP Targeting P2020RDB Board User Manual[M]. America: Freescale Semiconductor, 2010.

(上接第 153 页)

2004,20(2):68-69.

[2] 刘红彬,周 强. IP 网络摄像机及其在远程视频监控系统的 应用[J]. 设备管理与维修,2004,9(14):31-33.

[3] 刘红彬,周 强. IP 网络摄像机及其发展前景[J]. 煤矿现 代化,2005(4):44-46.

[4] 高 成. 基于移动网络的智能视频监控系统的设 计[D]. 长沙:国防科技大学,2010.

[5] 张 弘. 数字图像处理与分析[M]. 北京:机械工业出版 社,2007.

[6] Dirks B,Schimek M H,Verkuil H,et al. Video for Linux Two API Specification(Revision 0.24)[S]. 2008.

[7] 张 翔. 多功能 IP 摄像机的设计实现及流媒体传输的研 究[D]. 杭州:浙江大学,2008.

[8] Collins R,Lipton A J,Kanade T,et al. A system for video sur-veillance and monitoring; VSAM final report[R]. America: Carnegie Mellon University,2000;45-50.

[9] 侯宏录,李宁鸟,刘迪迪,等. 智能视频监控中运动目标检 测的研究[J]. 计算机技术与发展,2012,22(2):49-52.

[10] Yang Xingyu, Wu Haibin, Xu Songqing. Theapplication of GPRS wireless transmission in remote image monitored control system[J]. Microcomputer Information,2005(3):64-66.

基于P2020的SPI FLASH模式的研究与实现

作者: 王军, 胡明
作者单位: 电子科技大学 通信与信息工程学院, 四川 成都 611731
刊名: 计算机技术与发展
英文刊名: Computer Technology and Development
年, 卷(期): 2013(5)

本文链接: http://d.g.wanfangdata.com.cn/Periodical_wjtz201305042.aspx