RT Embedded http://www.kontronn.com

基于 MPC8548E 的固件设计

贾 亮[1]，岳 潇[1]，崔迎炜[2]
（1. 沈阳航空航天大学 电子信息工程学院，辽宁 沈阳 110136；2. 北京方天长久科技有限公司 北京 100084）

摘要：固件作为目标板启动最核心程序，不仅仅需要引导操作系统，更重要的是需要实现所有硬件的初始化和自检等，方便设备的维护和保养。U-Boot 作为一款通用的开源固件程序，具有良好的可移植性和完整的功能。通过分析 U-Boot 的目录结构和启动流程以及目标板的设计需求，来实现完成基于 MPC8548E 目标板的具体修改和移植，并增加关键硬件的自检功能，同时阐述通过设置环境变量来启动操作系统内核。对于不同的 CPU 和开发板，本 U-Boot 的启动原理分析和移植有一定的借鉴意义。

关键词：U-Boot；移植；MPC8548E；固件
中图分类号：TP311.1　　**文献标识码**：A　　**文章编号**：1674-6236(2012)21-0180-04

Firmware design based on MPC8548E

JIA Liang[1], YUE Xiao[1], CUI Ying-wei[2]
(1. *Institute of Electronic and Information Engineering, Shenyang Aerospace University, Shenyang* 110136, *China*;
2. *Beijing Fountain Microsystems Co.,Ltd., Beijing* 100084, *China*)

Abstract: The firmware which is a the core program of target board booting not only need to boot the operating system, more important is to implement all hardware initialization and self-checking, but also make device convenient equipment maintenance and repair. U-Boot which is a general firmware program with open source has the good portability and the complete function. After analysis the directory structure and the booting mechanism of U-Boot and design requirement of target board, the modification and porting in an embedded system board based on MPC8548E board are accomplished in particular, and it also can increase the self-check function of the key hardware. By setting up the environment variable to start operating system kernel. For different CPU and target board, the booting principle and transplanting of U-Boot have some reference significance.

Key words: U-Boot; porting; MPC8548E; firmware

U-Boot(Universal Boot Loader)是德国 DENX 小组开发的开放源码项目，它支持多种 CPU 体系结构的开发板，并且支持多种嵌入式操作系统内核，U-Boot 已经成为功能最多、灵活性最强的开放源码 BootLoader。作为一种通用的 Boot Loader，U-Boot 可以非常方便地移植到其他硬件平台上[1-2]。文中是基于 MPC8548E 的嵌入式目标板(即 FTC7110)和 U-Boot 源码资源来进行设计的，分析了 U-Boot 的目录结构和启动流程，同时对 U-Boot 源码设计以及设备自检程序的设计也做了相关的介绍，最后讲解如何通过设置环境变量来引导内核的启动。

1 开发板的硬件配置

FTC7110 板卡基于 mpc8548e 作为处理器，板上资源有 2 个网口、Nor Flash、Nand Flash、FPGA、CPLD、PHY 芯片 BCM5482s、DDR2、SRIO 交换机、PCI 桥片等。图 1 是本次设计的嵌入式目标板卡 FTC7110 的主要硬件资源结构框图。

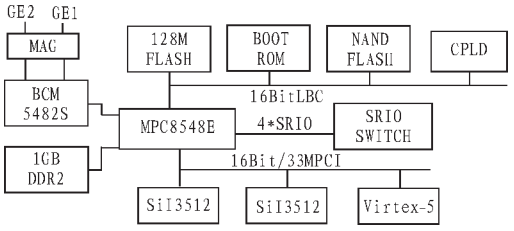

图 1　FTC7110 硬件结构框图

Fig. 1　Structure diagram of the hardware of FTC7110

2 U-Boot 目录结构

U-Boot 源程序包在顶层目录下一共有 17 个子目录，它们分别用于存放和管理不同的源程序。目录结构如下[3]：

api：存放 U-Boot 提供的用于拓展应用的相关接口函数；
arch：存放与各种 CPU 体系结构相关的代码；
board：存放基于各种开发板定制的代码；
common：存放通用的代码，提供相关支持的命令；
disk：存放磁盘分区的相关代码；

Creat Master PDF Editor

RT Embedded http://www.kontronn.com

doc:存放相关的阅读文档;

drivers:存放各种类型设备的驱动函数;

examples:存放示例程序;

fs:存放支持嵌入式开发板常见的文件系统;

include:存放基于各种定制板卡相关的头文件;

lib:存放通用的库文件;

nand_spl:存放 nand 存储器相关的代码;

net:存放网络相关的代码,小型的协议栈;

post:存放加点自检程序;

tools:存放用于编译和检查 U-Boot 目标文件;

3 U-Boot 启动流程分析

U-Boot 的启动大致可以分为两个阶段。第 1 个阶段在 Nor Flash 中执行,主要由汇编语言编写。第 2 阶段主要用 C 语言编写的,用于实现较复杂的硬件设备的初始化,同时也增加了代码的可移植性。

U-Boot 运行启动流程如图 2 所示,U-Boot 在复位启动后执行的第 1 个文件是 start.s,它位于 arch/powerpc/cpu/mpc85xx 中,接下来进行 U-Boot 启动的第 1 个阶段,完成一些针对 MPC8548E 的硬件初始化,主要包括初始化内核 e500、清楚指令 cache 和数据 cache、设置中断向量、清除和建立一些寄存器、设置 HID 寄存器、配置 MMU、初始化堆栈等,为系统创建一个有序可靠的初始环境,然后就进入到 U-Boot 启动的第 2 个阶段,首先进入第 1 个 C 语言函数 cpu_init_early_f 中,在这里主要是来初始化 TLB 和 LAW,为系统中的硬件设备来映射对应的地址空间。接着进入到第 2 个 C 语言函数 cpu_init_f 中,这里主要是对 TLB 和 LAW 进行详细的配置以实现相应的功能。之后进入比较重要的 C 语言函数 board_init_f 中,这里是对板上的硬件进行第 1 阶段初始化,主要由数组 init_sequence[]完成,该数组中的相关硬件初始化可以依据不同的目标板通过定义相关的宏定义来进行增加和删减。数组中的主要硬件初始化函数及函数功能如下:

probecpu:根据 cpu_type_list 识别 CPU 型号。

get_clocks:读取系统各种总线时钟频率,并保存于全局数据结构变量中。

init_timebase:重新初始化时基寄存器值。

env_init:初始化环境变量。

init_baudrate:初始化系统使用的波特率值。

checkboard:判断并打印开发板信息。

init_func_i2c:初始化 I2C 总线。

init_func_ram:初始化内存芯片配置。

在完成上述初始化后开始执行 relocate_code 函数,将代码从 Flash 中拷贝到 RAM 中,并记下当前执行代码的偏移,最后跳转到 RAM 中相应的位置继续执行板上硬件的第二阶段初始化,第二阶段是在 board_init_r 函数中执行,主要初始化包括有 PCI,PCIe,网口,Nor Flash,Nand Flash 等。在一切设备初始化完成之后,就进入函数 main_loop 中,等待 U-Boot 下命令的输入。

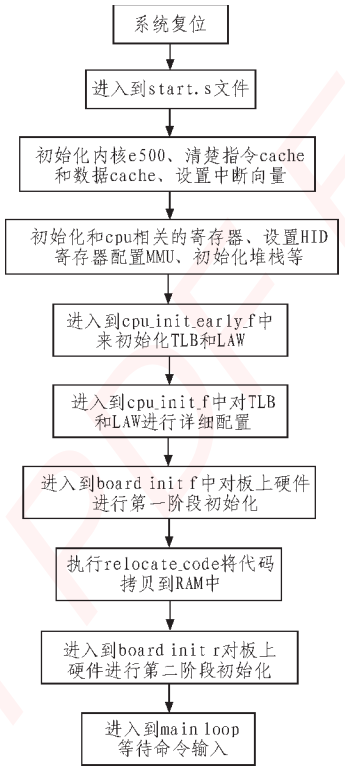

图 2 U-Boot 启动运行流程

Fig. 2 Booting mechanism of U-Boot

4 U-Boot 源码的设计及编译

U-Boot 中由于没有针对本目标板的源码,所以本次移植依据该源码的目录层次结构需要在 U-Boot 的 board 目录下为 FTC7110 嵌入式目标板新建 FTC 文件夹,添加 FTC 文件夹的目的是在该文件夹下可以建立基于 CPU 和硬件资源比较接近的 FTC 系列的其他板卡文件夹目录,所以这里需要在 FTC 文件夹下添加针对本次所采用的嵌入式板卡 FTC7110 文件夹,具体主要步骤如下:

1)添加 include/configs/FTC7110.h。这个文件是开发板的配置文件。它主要是添加定义开发板硬件资源的映射地址和大小以及添加针对目标板的资源信息的来进行增加,比如实现 MPC8548e 对 Nand Flash 的访问,这里需要添加对 MPC8548e 的 BR 和 OR 寄存器的配置[4]。

2)添加位于 board/FTC/FTC7110/下的 tlb.c 和 law.c。第 1 个 tlb.c 文件是设置开发板硬件资源对应的映射地址和映射大小。第 2 个 law.c 文件是控制是否使能访问这些映射的地址空间。

3)添加 board/FTC/FTC7110/FTC7110.c。该文件主要是完成对 FTC7110 目标板上相关硬件的初始化工作。

4)在 FTC 目录下添加 common 文件夹,该文件夹用于存放一些 FTC 系列板卡都支持的硬件配置代码或者是相关的硬件初始化代码,其主要目的是增加 U-Boot 源码的可扩展性。这里根据具体的实际需求,需要添加针对 MPC8548E 支

RT Embedded http://www.kontronn.com

持的 PCI、Nand Flash 和时钟等的相关配置代码。

5）U-Boot 源码不支持对本目标板 Nor Flash 的支持，需要自己添加编写针对该 Nor Flash 的驱动程序。

6）根据对板卡其他功能的设置和需求来对 U-Boot 源代码进行进一步的增加删除。

在进行完前五步后就已经可以进行编译，实现对 U-Boot 的启动。其中第 6 步主要是完善优化 U-Boot 的启动界面和根据实际的需求增加相应的功能和命令。截止到这里就已经实现了针对目标板资源的配置。这里编译的操作系统是 RHEL5.0，交叉编译环境是 ppc_85xx-gcc。编译 U-Boot 分为两步，第 1 步是配置编译的板卡，由于这里的 FTC7110 板卡目录是新增加的，所以需要在 U-Boot 根目录下的 boards.cfg 中添加相关配置文件以及在 FTC7110 和 common 文件夹下添加相应的 Makefile 文件，配置完成后在 U-Boot 源码根目录下输入 make FTC7110_config 命令。第 2 步编译，直接输入 make 命令即可。编译完成后在 U-Boot 顶层目录下会生成几个映像文件，这里通常采用将 u-boot.bin 二进制文件烧入到 Bootrom 中。目标板重新上电启动后，在超级终端下显示如下信息表示启动成功：

```
CPU:   8548E, Version: 2.0, (0x80390020)
Core:  E500, Version: 2.0, (0x80210020)
Clock Configuration:
       CPU0:1000 MHz,
       CCB:400  MHz,
       DDR:200  MHz (400 MT/s data rate), LBC:  50MHz
L1:    D-cache 32 kB enabled
       I-cache 32 kB enabled
Board: FTC-7110 6U CompactPCI SBC
      Beijing Fountain Microsystems Ltd. Co.
I2C:   ready
DDR: 512 MB (DDR2, 64-bit, CL=3, ECC on)
Flash: (Bank#1 - 512 KB, Bank#2 - 64 MB) 64.5 MB
L2:    512 KB enabled
NAND:  512 MiB
PCI1: 64 bit PCIX, <= 132 MHz, host, arbiter
In:    serial
Out:   serial
Err:   serial
Net:   eTSEC1 eTSEC2
FTC7110=>
```

5 设备自检程序的设计

针对本目标板的实际设计需求，对 Nor Flash、SRIO 总线等进行了自检程序的设计，确保各个设备和相关总线在 U-Boot 在启动后可以正常工作。

5.1 Nor Flash 自检程序设计

本目标板采用 Nor Flash 自检的原理是首先将 NorFlash 中的内容暂存到内存中去，接着依次对 Nor Flash 中的每个扇区采用 flash_write 接口函数向 Nor Flash 地址中写入固定值，然后再从该地址中读取，将读取的值与写入的固定值作比较，两值相等表示 Nor Flash 设备工作正常，否则表示异常，当然在向 Nor Flash 中写入数据之前须对 Nor Flash 进行擦除操作。最后将 Nor Flash 中原有内容再从内存中存入到 Nor Flash 中去。

如果该 Nor Flash 具有写保护功能时，在擦除 Flash 时，先检测写保护是否关闭。本目标板设计检测 Flash 写保护是通过读取 FPGA 的某一寄存器的值是否为 0 来进行判断，具体如下：

```
if (*(volatile unsigned  char *)( 0xee800012) == 0)
{
    printf("Flash is protected\n");
    return 0;
}
```

如果 Nor Flash 中已经存在内容，则需先将内容暂存到内存中，然后在一块内存区域中写入一个 Nor Flash 扇区大小的固定值来作为写入 Nor Flash 的值，采用 memset 函数实现，具体设计如下：

```
#define USER_FLASH_SECT_SIZE  0x10000
memset  ((unsigned  char  *)0x10000,  0x55,
USER_FLASH_SECT_SIZE)
```

上述完成之后采用 flash_erase 函数、flash_write 函数等来进行擦除写入验证该 Nor Flash 是否正常，具体设计如下：

```
rcode = flash_write  ((unsigned char *)0x10000, info->
start[i], USER_FLASH_SECT_SIZE);
testaddr = info->start[i];
while(testaddr<info->start[i]+USER_FLASH_SECT_SIZE)
{
val = *(volatile unsigned char *)testaddr;
if(0x55 == val){
    testaddr ++;
}
else{
    printf (" Flash Write Error!!! \n");
    return 1;
}
}
```

最后验证确定无误后，再将之前暂存到内存的内容存回到 Nor Flash 中去。

5.2 SRIO 自检程序设计

本目标板采用的 SRIO 交换机芯片是 IDT 公司的 TSI578。对于 SRIO 自检的设计，本设计采用的原理是通过配置 SRIO 总线来读取 TSI578 的 ID 号，以此来验证 SRIO 总线工作是否正常。

首先需要对 SRIO 总线进行必要的配置，通常分为 3 步：

Created in Master PDF Editor

RT Embedded http://www.kontronn.com

第 1 步配置 TLB;第 2 步配置 LAW;第 3 步配置 Maintenance Window[5-6]。由于前两步在 U-Boot 固件的时候已经做了相关的配置,所以这里只需对 Maintenance Window 进行相关的配置,具体设计如下:

```
*(unsigned long *)(0xef0d0c28) =0xa0000;
*(unsigned long *)(0xef0d0c20) =0x3fc00000;
*(unsigned long *)(0xef0d0c30) =0x80077015;
```

这里把通过 SRIO 总线对 TSI578 芯片的读操作叫做维护读。然后需要编写维护读的接口函数,这里采用宏定义的方法,函数名称为 MAINT_READ,具体设计如下:

```
#define MAINT_READ(device_id,hopcount,offset, ptrvalue)\
*(unsigned long *)(0xef0d0c20) = ((device_id) << 22)|((hopcount) << 12) | ((offset) >> 12); \
asm ("sync");\
ptrvalue = *((volatile unsigned long *) ((0xa0000000) + ((offset) & 0x3FFFFF)));\
asm ("sync");
```

最后调用 MAINT_READ 函数通过 SRIO 总线来读取 TSI578 设备的 ID 号,以此做出相应的判断,这里假定 TSI578 的 ID 号是 0x0578000d,具体如下:

```
MAINT_READ (0xFF, 0, 0, Reg);
if (Reg == 0x0578000d)
{
 printf ("SRIO MAINT_READ PASS\n");
}
```

6 操作系统的引导设计

U-Boot 修改配置的最终目的是引导 VxWorks 操作系统内核,由于该 VxWorks 操作系统内核文件是 ELF 格式,而对于 ELF 文件格式主要由 ELF Header、Program Header Table、Section Header Table 组成,所以本设计在 U-Boot 下采用 bootelf 命令来引导。

bootelf 命令首先检测该 ELF 文件是否有效,具体参考代码如下:

```
if (! valid_elf_image (addr))
{
printf(" ELF IMAGE IS Error!!! \n");
return 1;
}
```

对于上述 valid_elf_image 接口函数中的参数 addr 就是 VxWorks 系统内核所存放的具体地址。接着 bootelf 命令开始解析该 ELF 格式内核文件的 ELF Header、Section Header Table 和 Program Header Table 部分,这一部分主要通过 load_elf_image_phdr 和 load_elf_image_shdr 这两个接口函数来实现的。最后通过设置环境变量来引导启动该操作系统内核,假设这里通过网络下载到 Nor Flash 中然后来引导启动 vxworks.st 内核。具体输入以下命令:

```
FTC7110=> bootelf 0xf0000000"
```

0xf0000000 是本目标板的操作系统镜像文件的起始地址。

7 结束语

文中研究分析了将 U-Boot 成功移植到目标板 FTC7110 上,在 U-Boot 源码中添加 FTC7110 板卡的相关文件,同时该设计支持在线更新固件等功能,并在此基础上成功的引导了 VxWorks 操作系统,为后面的应用程序的开发奠定基础。对于不同的 CPU 和开发板,本 U-Boot 的启动原理分析和移植有一定的借鉴意义。

参考文献:

[1] 周庆松,史小军. U-Boot在AT91RM9200上的移植及启动分析[J]. 东南大学:电子科学与工程学院,2008:1-3.
ZHOU Qing-song,SHI Xiao-jun. Porting and start analysis of U-Boot based on AT91RM9200 board[J]. Southeast University:Department of Electronic Science and Engineering, 2008:1-3.

[2] 冯忠岭,童英华. ARM平台下U-Boot的移植[J].青海师范大学,2008:1-2.
FENG Zhong-ling,TONG Ying-hua. Transplanting of U-Boot under the ARM Platform[J]. Qinghai Normal University, 2008:1-2.

[3] 段薇. U-Boot在MPC8265平台上的移植与分析 [J]. 武汉船舶通信研究所,2010:2-3.
DUAN Wei. Transplanting and analysis of U-Boot on the MPC8265 platform[J]. Wuhan Maritime Communication Reseach Institute,2010:2-3.

[4] Freescale Semiconductor Inc.PowerPC MPC8548e reference manual[EB/OL].(2007-02-02).http://cache.freescale.com/files/32bit/doc/ref_manual/MPC8548ERM.pdf fsrch=1&sr=35.pdf.

[5] Freescale Semiconductor Inc. MPC8548E PowerQUICC III Integrated Processor Hardware Specifications [EB/OL]. (2012-09-02). http://cache.freescale.com/files/32bit/doc/data_sheet/MPC8548EEC.pdf pspll=1.pdf.

[6] Freescale Semiconductor Inc.RapidIO Bring-Up Procedure on PowerQUICC III reference manual[EB/OL]. (2004-01-11). http://cache.freescale.com/files/32bit/doc/app_note/AN2753.pdf fsrch=1&sr=2.pdf.

Created in Master PDF Editor

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组
5. 常用 GDB 命令中文速览

WeChat ID: kontronn

Created by Master PDF Editor

6. 嵌入式 C 进阶之道

Windows CE：

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]

PowerPC：

1. Freescale MPC8536 开发板原理图

ARM：

1. 基于 DiskOnChip 2000 的驱动程序设计及应用
2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码

Created in Master PDF Editor

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计

Created in Master PDF Editor