u-boot 学习指南

u-boot 是免费的，我们做嵌入式的一般只需要使用 u-boot 即可，但如果你想成为一个比较强的嵌入式系统工程师，而且还做了自己开发板，那么还是要学习一下如果将网上下载的通用 u-boot 移植到自己的开发板上，这个过程主要是修改主芯片相关代码以及开发板硬件相关代码，包括启动文件 Start.s、NAND 读写程序、USB 通信程序、相应的 IO 口配置等开发板上的资源，毕竟做 u-boot 这个开源项目的人不可能知道你用的是什么开发板，所以只能做个通用应用程序，然后再由做板的人去移植 u-boot。

我这里不讲移植，因为博芯已经为 SEP4020 和 MINI4020 移植好了 u-boot，其实移植 u-boot 也属于 BSP 的一部分，汤云峰和顾祥龙就是做这块的，我们只说如何使用 u-boot，或者说如何修改别人移植好的 u-boot 来满足自己的需求。

要修改一个程序（u-boot 也是一个程序，只不过非常大罢了），首先要搞懂一个程序的执行流程。对任何一个嵌入式项目工程而言，都会由以下几部分组成：

1. 启动文件：一般为 start.s 或者 startup.s，是在进入 main()函数之前对开发板的初始化程序，一般功能是初始化堆栈、开启 MMU、打开中断、初始化中断向量表等，这个在学习 ARM 编程后会有所了解，在启动文件的最后一句一般是挑战到__main 去执行，也即进入 main()函数。
2. 主函数(不一定是 main 函数)和子函数：这个不用说了。
3. 中断服务子程序：在 u-boot 中一般不使用中断，就目前为止我见到的 uboot 中都是没有中断的，所以这块也不需多说了。

因此我们只需要搞定启动文件和主函数就行了，两个文件的位置：cpu/arm920t/start.s，lib_arm/board.c。这里建议使用 SourceInsight 这款软件来看代码，因为 u-boot 中代码错综复杂，即使把执行流程全部给你写出来你也不一定能在整个 u-boot 文件夹中找到每个想要函数，而 SI 能帮你解决这个问题，只要选中某个函数，SI 就会自动的在整个文件中目录中搜索此函数的实体定义，并显示在屏幕下方的窗口中。以后在看内核源码、文件系统以及其它开源项目代码(madplay、web 服务器等)时，也尽量使用 SI，否则会无从下手。

1)、start.s

在 flash 中执行的引导代码，负责初始化硬件环境，把 u-boot 从 flash 加载到 RAM 中去，然后跳到 lib_arm/board.c 中的 start_armboot 中去执行。

1.1.6 版本的 start.s 流程：

1、硬件环境初始化：

进入 svc 模式；关闭 watch dog；屏蔽所有 IRQ 掩码；设置时钟频率 FCLK、HCLK、PCLK；清 I/D cache；禁止 MMU 和 CACHE；配置 memory control；

2、重定位：

如果当前代码不在链接指定的地址上（一般在编译源代码时会有“armlink --ro_base=0x50200000”这句，表示链接地址为 0x50200000）则需要把 u-boot 从当前位置拷贝到 RAM 指定位置中；

3、建立堆栈，堆栈是进入 C 函数前必须初始化的。

4、清.bss 区。

5、跳到 start_armboot 函数中执行。(lib_arm/board.c)

2)lib_arm/board.c:

start_armboot 是 U-Boot 执行的第一个 C 语言函数，我们平时一般第一个 C 函数是 main，这其实实在启动文件中的最后一句话中指定的，如果写为 main，就跳转到 main，写成 start_armboot 就跳转到 start_armboot。它主要的任务是完成系统初始化工作，进入主循环，

处理用户输入的命令。这里只简要列出了主要执行的函数流程：

```
void start_armboot (void)
{
  //全局数据变量指针 gd 占用 r8。
  DECLARE_GLOBAL_DATA_PTR;
  /* 给全局数据变量 gd 安排空间*/
  gd = (gd_t*)(_armboot_start - CFG_MALLOC_LEN - sizeof(gd_t));
  memset ((void*)gd, 0, sizeof (gd_t));
  /* 给板子数据变量 gd->bd 安排空间*/
  gd->bd = (bd_t*)((char*)gd - sizeof(bd_t));
  memset (gd->bd, 0, sizeof (bd_t));
  monitor_flash_len = _bss_start - _armboot_start;//取 u-boot 的长度。
  /* 顺序执行 init_sequence 数组中的初始化函数 */
  for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr)
  {
    if ((*init_fnc_ptr)() != 0)
    {
      hang ();
    }
  }
  /*配置可用的 Flash */
  size = flash_init ();
  ……
  /*初始化堆空间 */
  mem_malloc_init (_armboot_start - CFG_MALLOC_LEN);
  /* 重新定位环境变量，  */
  env_relocate ();
  /* 从环境变量中获取 IP 地址 */
  gd->bd->bi_ip_addr = getenv_IPaddr ("ipaddr");
  /* 以太网接口 MAC 地址 */
  ……
  devices_init (); /* 设备初始化 */
  jumptable_init (); //跳转表初始化
  console_init_r (); /* 完整地初始化控制台设备 */
  enable_interrupts (); /* 使能中断处理 */
  /* 通过环境变量初始化 */
  if ((s = getenv ("loadaddr")) != NULL)
  {
    load_addr = simple_strtoul (s, NULL, 16);
  }
  /* main_loop()循环不断执行 */
  for (;;)
  {
    main_loop (); /* 主循环函数处理执行用户命令 -- common/main.c */
```

```
    }
}
```

程序详解：

初始化函数序列 init_sequence[]，要学习执行大量函数时的这种写法，编写函数名数组，这对你们的编程能力有很大帮助。

init_sequence[]数组保存着基本的初始化函数指针。这些函数名称和实现的程序文件在下列注释中。

```
init_fnc_t *init_sequence[] = {
    cpu_init,           /* 基本的处理器相关配置 -- cpu/arm920t/cpu.c */
    board_init,   /* 基本的板级相关配置 -- board/smdk2410/smdk2410.c */
    interrupt_init,     /* 初始化例外处理 -- cpu/arm920t/s3c24x0/interrupt.c */
    env_init,       /* 初始化环境变量 -- common/env_flash.c */
    init_baudrate,  /* 初始化波特率设置 -- lib_arm/board.c */
    serial_init,    /* 串口通讯设置 -- cpu/arm920t/s3c24x0/serial.c */
    console_init_f,   /* 控制台初始化阶段 1 -- common/console.c */
    display_banner, /* 打印 u-boot 信息 -- lib_arm/board.c */
    dram_init,    /* 配置可用的 RAM -- board/smdk2410/smdk2410.c */
    display_dram_config, /* 显示 RAM 的配置大小 -- lib_arm/board.c */
    NULL,
};
```

整个 u-boot 的执行就进入等待用户输入命令，解析并执行命令的死循环中，也即 main_loop ()中。

3) common/main.c

main_loop()是在 common/main.c 中，可以在此函数中加入自己需要的模块初始化程序，比如液晶初始化、摄像头初始化、无线模块初始化等程序。

而且在 common/main.c 中还有一个比较重要的函数：abortboot()，它是用于设置启动延时的，是个回调函数，一般程序启动时需要快速按 space 键来进入 uboot 一键式菜单，这个功能就是这个函数实现的，可以根据需要修改此函数。

在 abortboot 中有这么几句：

```
    printf("Hit any key to stop autoboot: %2d ", bootdelay);
#endif

#if defined CONFIG_ZERO_BOOTDELAY_CHECK
    /*
     * Check if key already pressed
     * Don't check if bootdelay < 0
     */
    if (bootdelay >= 0) {
        if (tstc()) {   /* we got a key press   */
            (void) getc();  /* consume input    */
            puts ("\b\b\b 0");
            abort = 1;  /* don't auto boot  */
        }
    }
```

这段的意思是如果检测到键盘输入按键，则进入 u-boot 一键式菜单，否则启动系统。里面的 abort=1 很重要，这个全局变量为 1 时进入 u-boot 一键式菜单，为 0 时则直接引导启动 Linux 系统。如果希望每次都直接进入 u-boot 一键式菜单，可以用 abort=1 将这段替换掉。

再说一个比较重要的地方：4020 的 u-boot 把一键式菜单给去掉了，但我们可以再添上，可以在 main_loop()里的注释“Main Loop for Monitor Command Processing”处可以添加一键式菜单，如下图所示：

```
#endif /* CONFIG_MENUKEY */
#endif  /* CONFIG_BOOTDELAY */

#ifdef CONFIG_AMIGAONEG3SE
    {
        extern void video_banner(void);
        video_banner();
    }
#endif

    /*
     * Main Loop for Monitor Command Processing
     */
#ifdef CFG_HUSH_PARSER
    parse_file_outer();
    /* This point is never reached */
    for (;;);
#else
```

因为一次 main_loop()可以执行一次 u-boot 命令，因此我们可以在此处添加如下代码（这里的 printf 可以将信息打印至串口，因为这是在开发板上运行的，而不是平时在 linux 中的，因此控制台变成串口了）：

```
printf("\r\n###############     Main Menu     ###############\r\n ");
printf("\r\n  *********    EmbedRoad Studio   *********\r\n \r\n ");
printf("[f] Format the Nand Flash\r\n");
printf("[0] Set the boot parameters\r\n ");
printf("[1] Download u-boot to Nand Flash\r\n ");
printf("[2] Download FontLibrary to Nand Flash\r\n ");
printf("[3] Download LOGO Picture (.bin) to Nand Flash\r\n ");
printf("[4] Download Program to SDRAM and Run it\r\n");
printf("[d] Download User Program\r\n n");
printf("[e] Exec User Program\r\n ");
printf("[q] Quit from menu\r\n ");
printf("\r\n Enter your selection: ");
while(!(((keyselect >= '0') && (keyselect <= '4')) ||
            ((keyselect == 'f') || (keyselect == 'F')) ||
           ((keyselect == 'q') || (keyselect == 'Q')) ||
           ((keyselect == 'd') || (keyselect == 'D')) ||
           ((keyselect == 'e') || (keyselect == 'E')) ))
{
     keyselect = serial_getc();
}
switch (keyselect)
{
     case '0':
          break;
     case '1':
          break;
     case '2':
          break;
               .
               .
               .
     default : break;
```

```
}
```

可以在 switch 语句的相应地方写上面菜单中对应的命令，一般比较常用的命令有：

1. bootm：
2. cp：
3. echo
4. nand erase：
5. nand write：
6. nand read：
7. go：
8. loadb：
9. loads：
10. nfs
11. setenv
12. printenv
13. sleep
14. tftpboot

这几个命令在快盘中的“U-Boot 的常用命令.pdf”详细讲了，而且在本文下面的代码中也会提及具体使用方式。如果你自己的菜单过于庞大，尽量在 common 文件夹中新建一个 c 文件用来专门实现一键式菜单，像三星的板子配套的 uboot 一般有一个 cmd_menu.c 文件，也是在 common/中，当然添加一个 C 文件后不要忘记在编译前在 makefile 中把这个新添加的文件包含进去。

那么对于上面写的那个一键式菜单的实现代码，也即 switch (keyselect)中的代码可以写成：

```
switch (keyselect)
{
    case '0':  //cfg parameter
    {
        param_menu_shell();
        break;
    }
    case '1':  //Download u-boot to Nand Flash
    {
        strcpy(cmd_buf, "dnw 0xc0000000; nand erase 0x0 0x40000; nand write.jffs2
0xc0000000 0x0 $(filesize)");
        run_command(cmd_buf, 0);
        break;
    }
    case '2':   //FontLibrary
    {
        sprintf(cmd_buf, "dnw 0xc0000000; nand erase 0x80000 0x50000; nand
    write.jffs2 0xc0000000 0x80000 0x50000");
        run_command(cmd_buf, 0);
        break;
    }
    case '3':  //Pitcure
```

```
                {
                }
                case '4':  //Download Program to SDRAM and Run it
                {
                    char addr_buff[12]="W0";
                    printf("Enter download address:(eg: 0xc0000000)WrWn+");
                    readline(NULL);
                    strcpy(addr_buff,console_buffer);
                    sprintf(cmd_buf, "dnw %s;go %s", addr_buff, addr_buff);
                    run_command(cmd_buf, 0);
                    break;
                }
                case 'f':  //Format the Nand Flash
                case 'F':
                {
                    strcpy(cmd_buf, "nand scrub ");
                    run_command(cmd_buf, 0);
                    break;
                }

                case 'D':  //Download User Program
                case 'd':
                {
                    strcpy(cmd_buf, "dnw 0xc0000000; nand erase 0x2100000 0x40000; nand
write.jffs2 0xc0000000 0x2100000 0x40000");
                    run_command(cmd_buf, 0);
                }
                case 'E':
                case 'e':
                {
                    strcpy(cmd_buf, "nand read 0x53000000 0x3000000 0x80000; go 0x53000000");
                    run_command(cmd_buf, 0);
                }
                case 'Q':
                case 'q':
                {
                    return;
                    break;
                }
            }
```

可以这么说，你平时在 u-boot 下手动输入的哪些指令都是可以写在 u-boot 程序中的，这就是我们要修改 u-boot 的目的，既然做一个项目，肯定不可能总是希望用户把 uboot 的命令也全记住，所以要自己编写一键式菜单，方便用户使用，如上面列出代码所示。稍微讲解一下这段代码中的命令：

1、dnw 0xc0000000：

dnw 命令是三星公司自己定义的一个命令，主要是用于通过 usb、com 口从 PC 机下载程序到内存的。这句话的意思是等待 PC 传送.bin 文件，然后把接收的文件存放在 0xc0000000 处。我们也可以定义自己的命令的，而且很简单，有兴趣的到网上查找制作自己命令的教程。

2、nand erase 0x2100000 0x40000

nand erase 是擦除 NAND Flash 中某段空间，这句话的意思是从 NAND 的偏移地址 0x2100000 开始，擦除 0x40000 大小。如果 NAND 在 ARM 芯片里的存储器空间基址为 0x40000000，那么擦除的区域为 0x42100000~0x42140000。

3、nand read 0x53000000 0x3000000 0x80000

nand read 可以从 NAND Flash 中读取一段数据并存放在 SDRAM(内存)中，这句命令的意思是从 NAND 的偏移地址 0x3000000(也即 NAND_BASE+0x3000000)开始读取 0x80000 大小的数据并存放在 0x53000000 处。

4、nand write.jffs2 0xc0000000 0x2100000 0x40000

nand write.jffs2 可以将内存里的一段数据写到 NAND 中，也即保存数据（内存是易失性存储器，而 NAND 可以长久保存数据）。这个命令的意思是将内存 0xc0000000 处的 0x40000 大小的数据烧写到 NAND 的偏移 0x2100000 去，也即 NAND_BASE+0x2100000。

5、go 0x53000000

表示使程序指针跳转到内存 0x53000000 处，开始执行这地方的代码。使用这个命令我们可以做出很多有趣的功能，比如在 NAND 中预先保存四个程序，然后通过按不同的按键去引导不同的程序，也即所谓的多系统启动，学了这些之后你会发现很多东西原来这么简单，其执行过程是：

当按某一个键是，u-boot 理解执行以下这个命令。注意：多个命令之间用分号隔开，末尾不加分号：

nand read 0x53000000 0x3000000 0x80000; go 0x53000000

也即先从 NAND 中把程序拷贝到内存某个地址，然后跳转到内存的那个地址。

-- 六度空间

2010/11/3