DJYOS

Iboot用户手册

目录

1. 概述 4

2. IBOOT 使用 4

2.1. IBOOT 固化 4
2.2. IBOOT 典型资源占用 5
2.3. 用 YMODEM 更新 APP 5
2.3.1. 工具准备 5
2.3.2. 升级步骤 5
2.4. FTP 更新 APP 6
2.4.1. 工具准备 6
2.4.2. 升级步骤 6
2.5. 强制启动 IBOOT 7
2.6. IAP 相关 SHELL 命令 7
2.6.1. runiboot：运行 iboot 7
2.6.2. runiapp：运行 app 8
2.6.3. appinfo：获取 app bin 文件信息 8
2.6.4. iapstatus：获取 iboot 状态信息 8
2.6.5. iapver：获取 IAP 版本信息 8
2.6.6. iapmode：获取 IAP 运行模式 9

3. IBOOT 原理 9

3.1. APP 与 IBOOT 之间的关联 9
3.2. APP 启动方式 9
3.2.1. MCU 模式 9
3.2.2. CPU 模式 10
3.3. 下载 APP 11
3.4. 快速启动应用程序 12

4. 无 IBOOT 模式 13

5. IBOOT 移植 13

5.1. 直接使用 SDK 包中的 IBOOT 13
5.2. 从 SDK 包的 IBOOT 中修改 14
5.3. IBOOT 配置 14
5.4. YMODEM 配置 15
5.5. TCPIP 配置 15
5.6. LDS 文件编写 15
5.6.1. MCU 模式 15
5.6.2. CPU 模式 16
5.7. 通信方式 16
5.7.1. Uart 16

5.7.2. 网络 17
5.7.3. 其他 18
5.8. IAPFS 驱动 18
5.9. 专用硬件 18
6. APP 开发调试 18
7. API 接口函数 19
7.1. MODULEINSTALL_IAP：IAP 模块加载 19
7.2. IAP_GETAPPSTARTADDR：获取 APP 起始地址 19
7.3. IAP_GETAPPSIZE：获取 APP 大小 19
7.4. IAP_GETAPPCRC：获取 APP 的 CRC 值 19
7.5. IAP_GETAPPSTARTCODERAMADDR：获取加载到 RAM 代码起始地址 20
7.6. IAP_GETAPPCODERAMSIZE：获取加载到 RAM 代码尺寸 20
7.7. IAP_LOADAPPFROMFILE：从文件加载 APP 20
8. 附录一：超级终端安装 21
9. 附录二：FTP 安装 21

DJYOS Iboot 用户手册

编写：贺敏 2016 年 10 月 24 日
Review：_年_月_日
审阅:罗侍田

1. 概述

本文描述的是 Iboot V1.0，只针对 SI 模式，即 djyos 和 APP 编译在一个 bin 文件中加载运行的模式，Iboot 的 IAP 功能，也是针对该 bin 文件进行的。该 bin 文件可能直接烧录在 mcu 的片内 flash 中，也可能保存在文件系统中。

DJYOS 的启动加载器命名为 Iboot（即 IAP BOOT），Iboot 与传统 boot 程序相比，强化了 IAP 功能，故名。

Iboot 主要功能有：

1、 初始化最小系统硬件，包括 CPU 核心寄存器初始化、PLL、内存参数配置等。

2、 加载和运行应用程序，加强了 APP 有效性和完整性的检查，杜绝因 APP 错误导致设备变成砖头的风险；

3、 在线下载 APP，下载方式可灵活选择各种通信口，可用 ymodem 和 FTP。官方提供的 SDK 包中，都包含了（uart + ymodem）下载模式，如果有网口的平台，还包括了 FTP 模式。

Iboot 强化了 IAP 功能，升级过程中，通信错误、PC 死机、断电、接触不良，都可能导致升级失败。升级失败的后果，往往导致引导程序和应用程序都无法正常运行，甚至硬件仿真器都无法连接，即所谓“变砖头”。DJYOS 的 IAP 模块成功解决了代码升级过程中失败所造成的尴尬局面，更加符合嵌入式应用“高可靠”的特点。

DJYOS 在线应用编程模块的特点包括：

- 灵活的在 Iboot 和 APP 之间进行切换；
- Iboot 所需的 ROM 和 RAM 空间小；
- 支持 ymodem、FTP 两种协议，多种通信方式（IIC、SPI、uart、网口、无线等）；
- 强校验机制，防止升级过程中失败“变砖”的尴尬境地；
- 强制运行 Iboot 机制，防止 APP 本身存在 bug 导致无法运行的问题；

注意区别 Iboot 的 IAP 功能和 cpu 硬件自带的 IAP 功能。有些 CPU，例如 STM32，在 CPU 内嵌入了一小片 ROM，固化了一段用于下载用户程序的代码，也称为 IAP。Iboot 的 IAP 是通用的，不管哪家的 cpu，都一样；而厂家的 IAP 是专用的，每家的实现方法都不一样。

2. Iboot 使用

2.1. Iboot 固化

Iboot 是 CPU 启动后，运行的第一个程序，它是固化在 flash 中的。

注意：如果不先烧录 Iboot，debug.bin 和 release.bin 即使烧录了，也无法运行的，更无法调试。Iboot 固化到 0 地址，debug.bin 固化地址参考 memory.lds，release.bin 必须由 Iboot 下载。

固化 Iboot 前，CPU 里面没有任何用户程序，固化 Iboot 的方法不外乎有 3 种：

1、 离线烧录，即芯片焊接之前先烧录好，或者有些支持 SD 卡启动的系统，用 PC 机烧录。

2、 用硬件仿真器在线烧录。

3、 用厂家的 IAP 功能在线烧录。

具体烧录方法与硬件相关，在此难于一一罗列，请参考你的硬件设计。

2.2. Iboot 典型资源占用

APP 运行时，Iboot 已经退出，故 Iboot 不占用系统内存。

Iboot 固化在 flash 中，须占用一定的 flash 空间，几种典型配置的占用如下。

如表 2-1 所示，举例说明了 DJYOS 的 IAP 模块的 Iboot 消耗 CPU 资源，供移植应用参考。

表 2-1 Iboot 所需资源表

芯片	通信模块	Iboot 所需资源
LPC1759(Cortex-M3)	串口	ROM < 75KB，RAM < 32KB
MK70FN1M(Cortex-M4)	串口+ FTP	ROM < 200KB，RAM < 100KB(包含网卡驱动发送接收缓冲 10KB)
ATSAME70Q21(Cortex-M7)	串口+ FTP	ROM < 225KB，RAM < 128KB(包含网卡驱动发送接收缓冲区 18KB)

2.3. 用 Ymodem 更新 APP

2.3.1. 工具准备

1. 串口线（收发地三线即可），可用 USB 转串口线。
2. 安装超级终端工具，安装方法参看附录一：超级终端安装。注意，xp 及以下版本的超级终端传输速度很慢（约 3KBps 以下），win7 及以上版本可达 10KBps。

2.3.2. 升级步骤

1、 连接好串口线，确保硬件正确连通，终端和电脑的串口收发交叉连接。

2、 设置串口为：baud=115200，N、8、1，无流控。

3、 打开并连接超级终端，在键盘上点击“Enter”会在超级终端上输出”/>”符号表示已经成功连接上，如果未输出上述符号，说明未连上，需要检查串口驱动是否安装好，参数设置是否正确，电脑的端口号是否匹配，可尝试换个端口试用。

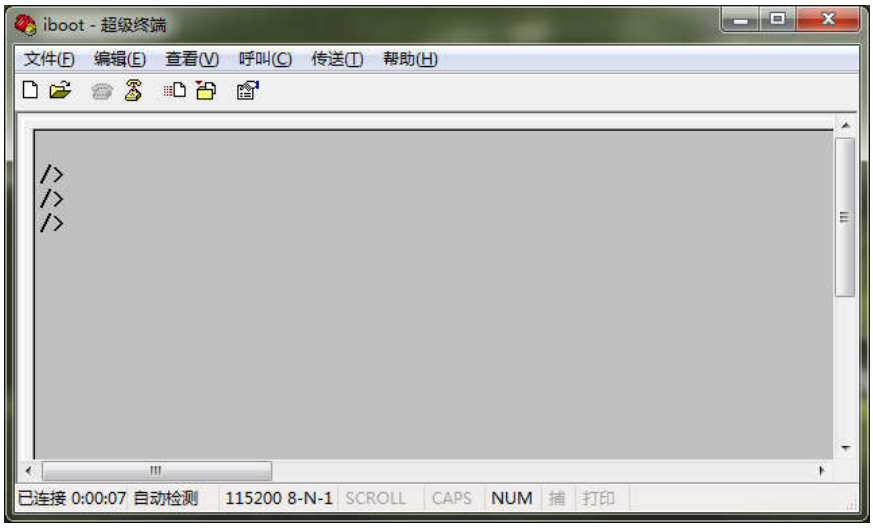

图 2-1 超级终端配置 6

4、 在超级终端输入”iapmode”+Enter 键后，查看板件当前运行模式，只有两种运行模式“Iboot/APP”，如果运行模式为 APP，则需要输入”runiboot”，这时板件会软重启，等到软重启完毕，重新输入”iapmode”+Enter，会在超级终端显示“Run Mode:Iboot”，这时表明板件运行在 Iboot 模式下。只有在 Iboot 模式下才可升级应用程序。

5、 在超级终端上输入“download”命令+Enter 键后，会显示 60s 倒计时，在倒计时时间

内需要完成以下步骤 7：

6、 点击上图红色小图标，会弹出发送文件对话框，在协议项选择 Ymodem，在文件名选择需要升级的 bin 文件，如下图所示，点击发送，即可开启升级过程。

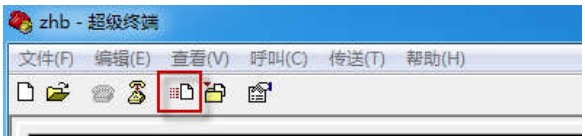

图 2-2 超级终端配置 6

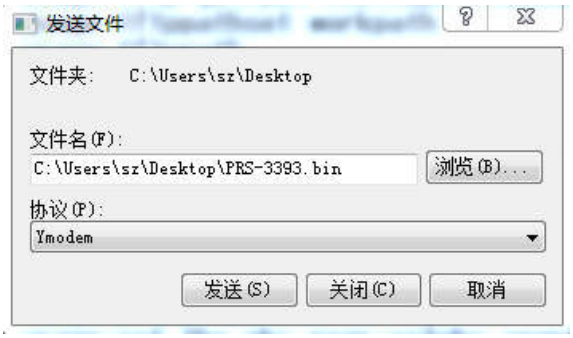

图 2-3 超级终端配置 7

7、 正常升级过程如图所示，会实时显示升级进度，升级完成后会在超级终端显示“YMODEM SUCCRSSED!”

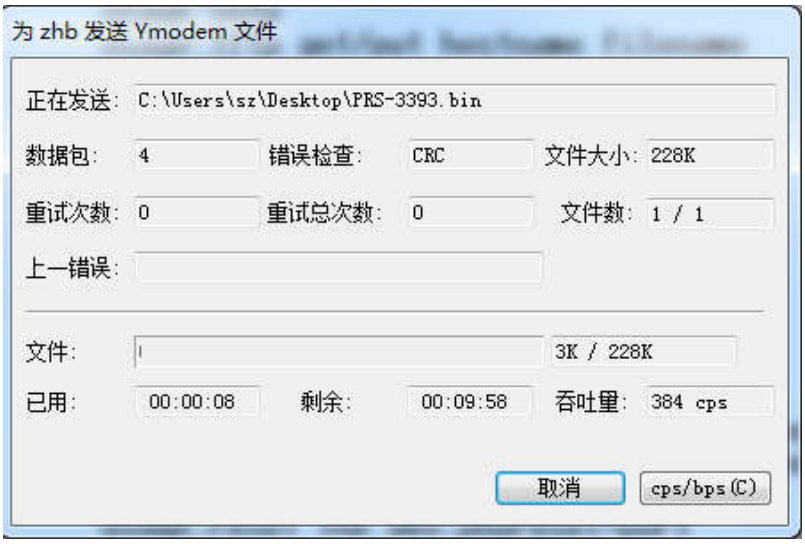

图 2-4 超级终端配置 8

8、 在超级终端输入”runapp”+ Enter 键后，即可使板件运行新升级的应用程序，升级即可大功告成。

2.4. ftp 更新 APP

2.4.1. 工具准备

1. 在电脑上安装 ftp 工具，安装方法参看附录二：FTP 安装。
2. 一根可正常使用的网线。
3. 准备好待升级的应用程序的 bin 文件。

2.4.2. 升级步骤

1. 将本机 IPV4 地址改到设备同一网段(官方 SDK 默认配置为 192.168.0.179)，例如 192.168.0.100。
2. 将网线一端接板件对应网口。
3. ping 设备，确认是否能 ping 通，若不通检查网线是否可用。
4. ping 通后打开 FlashFXP 工具，点击快速连接，配置如下，密码为 123456。

图 2-5 FTP 配置图 1

5. 点“连接”按钮之后会在左上侧看到如下图目录。

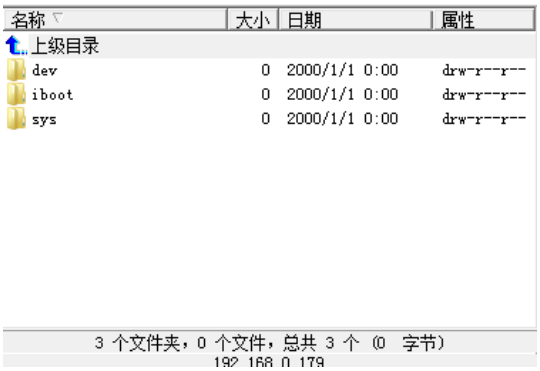

图 2-6 显示 Iboot 目录

6. 选中待升级的 bin 文件，传送至 iboot 目录即可。
7. ftp 传送完毕后，可通过在串口终端上输入”runapp”命令或者断电重启，即可完成升级。

2.5. 强制启动 Iboot

本功能须硬件设计支持。

应用程序错误（开发中比较常见），应用现场下错程序（下程序时选择了错误的文件），或者 flash 损坏（产品运行过程中的小概率事件），会使设备无法启动。甚至有些错误会导致硬件仿真器也无法连接设备的情况，只能眼睁睁地看着设备变成砖头。况且，应用现场连接硬仿真器也不现实。

Djyos 考虑到这种情况，允许用户设计一个专用硬件来解决这个问题。一般可用一个 gpio 口，用跳线控制，或者由板子上另一颗 CPU 控制。跳线跳上，则复位后强制运行 Iboot。

此外，异常组件在处理异常处理行动时，可配置为 EN_EXP_DEAL_RUNIBOOT，应用程序发生异常时，可使程序重新恢复到 iboot 程序运行。

2.6. IAP 相关 shell 命令

2.6.1. runiboot：运行 iboot

用法：

/>runiboot

参数：无。

说明：

从当前运行模式切换到 iboot 模式。

2.6.2. runiapp：运行 app

用法：

/>runiapp

参数：无。

说明：

从当前运行模式切换到 app 模式。

2.6.3. appinfo：获取 app bin 文件信息

用法：

/>appinfo

参数：无。

说明：

获取应用程序 bin 文件的信息，信息包含文件名、文件大小和 32 位的 CRC 值。

2.6.4. iapstatus：获取 iboot 状态信息

用法：

/>iapstatus

参数：无。

说明：

获取 iboot 状态信息，iboot 的状态信息包含在枚举量_IBOOT_ERR_STATUS_。

```
enum _IBOOT_ERR_STATUS_
{
    EN_NO_ERR=0,
    EN_FORCE_IBOOT,
    EN_RAM_IBOOT_FLAG,
    EN_LOAD_FROM_DATA_MODE,
    EN_CRC_ERR,
    EN_APP_FLAG_ERR,
    EN_FILE_NO_EXSIT_ERR,
    EN_FILE_SIZE_INVALID_ERR,
    EN_BIN_INCOMPLETED_ERR,
    EN_LDS_MISMATCH,
};
```

EN_NO_ERR：正常使用 iboot，无错误。

EN_FORCE_IBOOT：硬件强制运行 iboot。

EN_RAM_IBOOT_FLAG：shell 修改内存标记，运行 iboot。

EN_LOAD_FROM_DATA_MODE：应用程序从数据存储器加载代码。

EN_CRC_ERR：应用程序 CRC 校验错误。

EN_APP_FLAG_ERR：应用程序“release”标记错误。

EN_FILE_NO_EXIST_ERR：应用程序 bin 文件不存在错误。

EN_FILE_SIZE_INVALID_ERR：应用程序 bin 文件大小无效错误。

EN_BIN_INCOMPLETED_ERR：应用程序 bin 文件不完整错误。

EN_LDS_MISMATCH：应用程序 bin 存储地址错误。

2.6.5. iapver：获取 IAP 版本信息

用法：

/>iapver

参数：无。

说明：

获取 IAP 版本信息，IAP 的版本信息与内核版本信息一致，例如“Version:djyosV1.2.0-Nov 17 2016-11:30:30”。

2.6.6. iapmode：获取 IAP 运行模式

用法：

/>iapmode

参数：无。

说明：

获取 IAP 运行模式，运行模式分为 iboot 和 APP 两种。

3. Iboot 原理

Iboot 本身就是一个 djyos 的应用程序。

Iboot 的核心功能有：初始化 CPU、启动加载用户程序、在线下载用户程序。

“初始化 CPU”，即为用户应用程序（Iboot 本身也是应用程序）提供最基本的运行环境，主要完成的工作有：CPU 核心寄存器初始化、PLL、内存参数配置，与具体硬件相关，这里就不赘述了。Djyos 的源码目录中，“bsp\startup\”目录下，包含了许多开发板的启动代码，大家可以参考。

3.1. APP 与 Iboot 之间的关联

Iboot 工程的 memory.lds 文件中，指定了 APP 的起始地址，APP 的 memory.lds 文件中，也要指定自己的起始地址，两者必须一致，即 memory.lds 文件中的 IbootSize 及定义必须一致。

PS：IbootSize 如果太小，编译 Iboot 时会报错的。

Iboot 和 APP 中 IbootSize 设置不一致的话，启动时 shell 会输出错误。

关于 lds 文件，在第 5.6 节有详细说明。

3.2. APP 启动方式

3.2.1. MCU 模式

处理器内有一片尺寸足够的 flash（XIP）和 ram，指令和只读数据在片内 flash 中，ram 做运行内存。即使有外接存储器，也主要做辅助用，例如用来保存图片、字库、记录等，或者做通信缓冲区。

也有一些比较特殊，片内不带 flash，而是在片外挂一片 SPI 接口的 flash，这些 mcu 内部做了转换，映射到内存地址，使之能够执行程序，例如 ESP8266。此类接口的存储器，访问会有较大延迟，连续地址访问速度较快，但如果读写地址发生跳变，第一个字节访问就会插入额外延迟。

APP 和 Iboot 都保存并运行在片内 flash 和片内 ram 中。Iboot 要占用一部分 flash，由 lds 文件控制 Iboot 和 APP 分别保存位置。

CPU 初始化代码放在 Iboot 工程中，并编译到 iboot.bin 中，完成 CPU 初始化后，将检查 APP 的完整性，校验正确性（由 g_IbootCRC 配置决定是否执行），而后，直接决定加载执行 APP 还是 Iboot。如果要执行 APP，则 Iboot 本身并不加载运行，以加快启动速度。

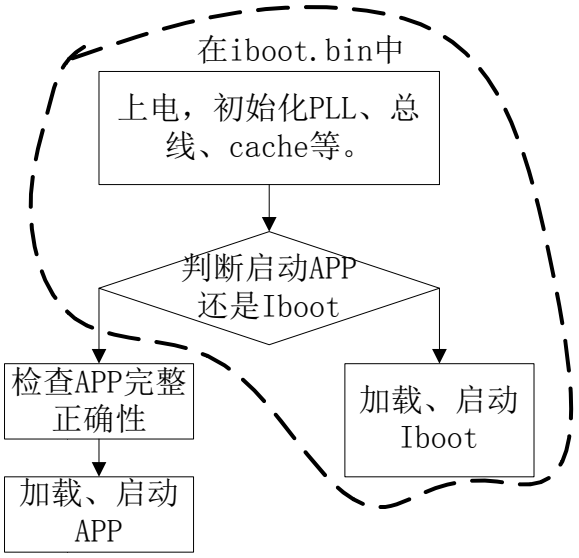

图 3-1 MCU 模式启动示意图

3.2.2. CPU 模式

CPU 模式有比较大的、速度较快的 RAM，程序一般在 ram 中运行。

而 ROM 的情况，比较复杂，典型的情况有：

1、 有一片较大的、足够保存全部程序（APP+Iboot）的 norflash，该 flash 可以执行程序，但速度较慢。

2、 大容量、不能执行程序的主存，外加较小的可以执行程序的存储器，可能是 norflash，也可能是 ram。如果是 ram，硬件会自动从主存 copy 一段程序到 ram 中。

3.2.2.1. 裸数据模式

这是 CPU 上跑 rtos 最常见的模式，Iboot 和 APP 都直接存储在主存的指定位置，APP 需要搬移到 ram 中运行。如果有足够保存 Iboot 的、可执行代码的 norflash，则 Iboot 不需要搬移到 ram 中，否则也要搬。

Iboot 需要把 APP 从主存中搬移到 ram 中，再跳转到 ram 中执行。

虚线框的含义，是可能需要，也可能不需要这个过程。

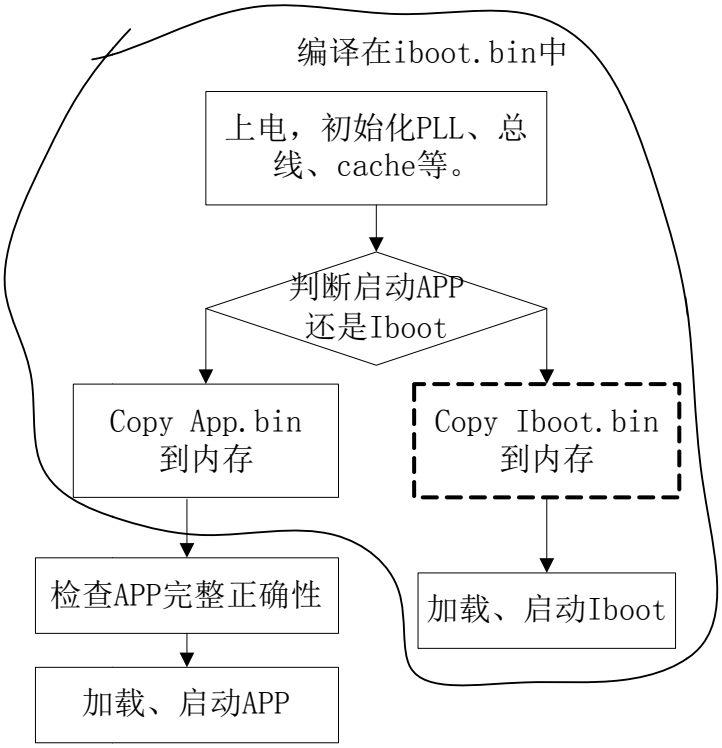

图 3-2 裸数据模式启动示意图

3.2.2.2. 文件系统模式

主存按照文件系统组织，APP 以文件的形式保存。Iboot 则分两种情况，一是直接保存在可执行代码的 flash 中，要求该 flash 应该足够大，在 256K 及以上；二是直接存在主存指定位置（文件系统应该空出这段空间），上电或复位时，由硬件或一小段引导程序加载到内存中运行。

Iboot 必须包含文件系统，本模式下，APP 不能直接启动，而是必须等 Iboot 完全启动后，从文件系统中加载再启动。

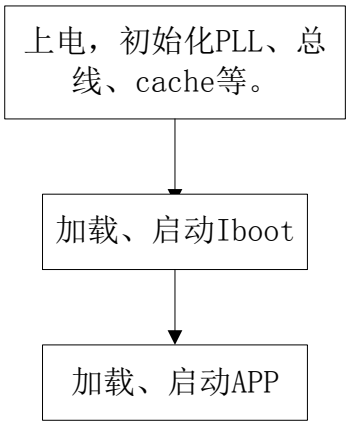

图 3-3 文件系统模式启动示意图

3.3. 下载 APP

与 IAP 功能完美结合是 Iboot 的特色，IAP 的核心功能是下载 APP 到目标机中，同时，还要写入程序完整性标记和校验 CRC，用于启动 APP 前检查 APP 是否完整正确。

如图 3-4 所示，Iboot 支持多种数据接口，官方提供的 sdk 包中，已经提供了 uart 和 ftp（如果

有网口）下载模式。如果用户要改变下载程序的通信口，无论使用哪种接口，除网口外，都推荐使用 ymodem 协议，并按照 djyos 的 driver 框架来编写驱动，然后在 ymodemcfg.c 中，配置设备名，如果不配置，则使用默认设备名 stdin。

下载端口，既可以使用 shell 端口（例如 uart），也支持专用端口，如果共用端口，则在数据传输期间，拒绝接收 shell 命令。

不同型号的 mcu，其 flash 组织和写入形式千差万别，外接 flash 的形式更是多种多样。实际应用中，有直接写入 flash 指定位置的，也有使用文件系统的。无论哪种 flash，无论怎样组织数据，Iboot 都使用文件系统接口，ymodem 和 FTP 收到数据后，按照标准的文件系统 API 调用来写入文件即可。如果是 mcu 的片内 flash，djyos 提供了 IAPFS 文件系统（与 fat32 一样，是一种文件系统实现），如果用户改变了 mcu 型号，就要依照第 4 章所述的方法，编写必要的接口函数。如果是 nor/nand flash，情况稍复杂，如果用的是裸数据模式（见 3.2.2.1 节），也使用 BootFS 文件系统；否则，依你选择的具体文件系统实现而定。

这样，虽然我们面对的是各种各样的通信口，各种各样的存储介质，但 Iboot 是相同的，不同的只是少数接口函数而已。

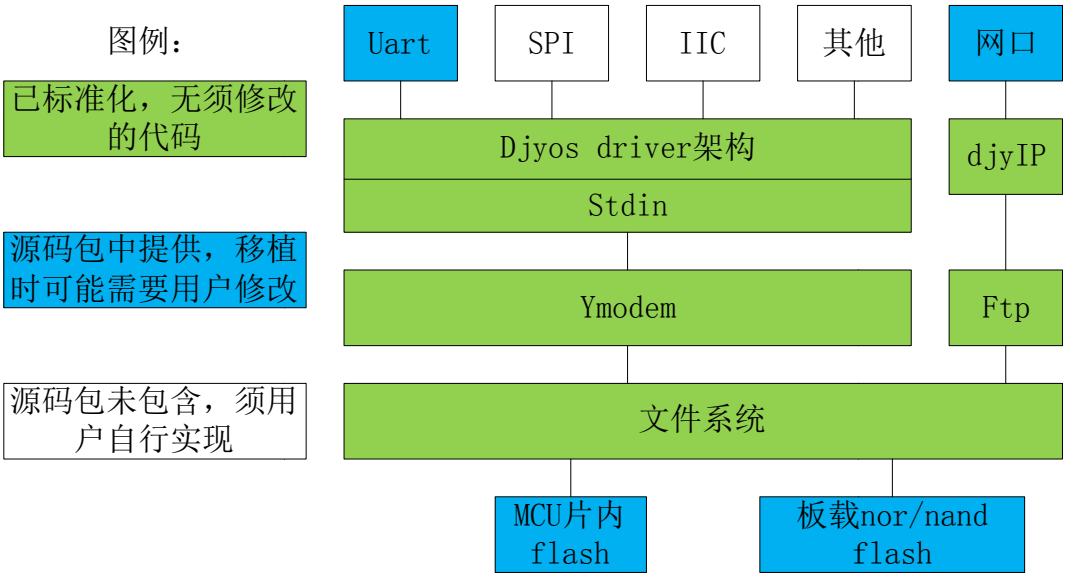

图 3-4 Iboot 下载 APP

3.4. 快速启动应用程序

传统 bootloader 一般要完整启动 bootloader 本身后，再启动和加载应用程序；Iboot 不同，从图 3-1 和图 3-2 中可见，无论是 MCU 模式，还是裸数据模式，初始化后，APP 都是直接启动的。唯一需要额外花销的时间，就是判断启动 APP 还是启动 Iboot，大大加快应用程序的启动速度。

启动速度重要吗？

很多人认为，启动速度不要紧，秉承这个理念。

Dos 的启动是秒级的。

Windows 时代，迅速发展到分钟级。

每次开机，用户都得傻傻地等待。

知道 SSD，才回到 10 秒级别。

从微软到 Google，又有了新发展。

纯 SSD 的 Android 手机，启动时间也是分钟级的。

上面说的，是消费品而已，顶多让用户等待一下。

对于嵌入式产品的启动时间呢？问问元芳怎么看。

“大人，启动慢一点，就像晚一些按开机按钮，关系不大”。

“那如果正在运行的产品崩溃重启呢？”

“大人，这就坏了，启动慢会导致设备长时间失去功能，特别是关键控制岗位，可能是致命的。”

4. 无 Iboot 模式

从图 3-1 中可见，djysdk 编译出来的 APP 中，无论 debug 还是 release，都不包含 CPU 初始化代码，是必须在 Iboot 支持下才能运行的。

但有些用户，出于成本考虑或历史传统，CPU 的 flash size 非常非常小，无法同时容纳 Iboot 和 APP。怎么办?

我们知道，Iboot 本身就是 djyos 应用程序，故只要把用户代码，直接编译到 Iboot 中，Iboot 就成了 APP。

方法如下：

1、 下载 djysdk。

2、 在 eclipse 中 import Iboot project

3、 直接在相应板件的源码中，依下图修改。如何在 Iboot 中添加自己的板件，参见第 5.2 节。

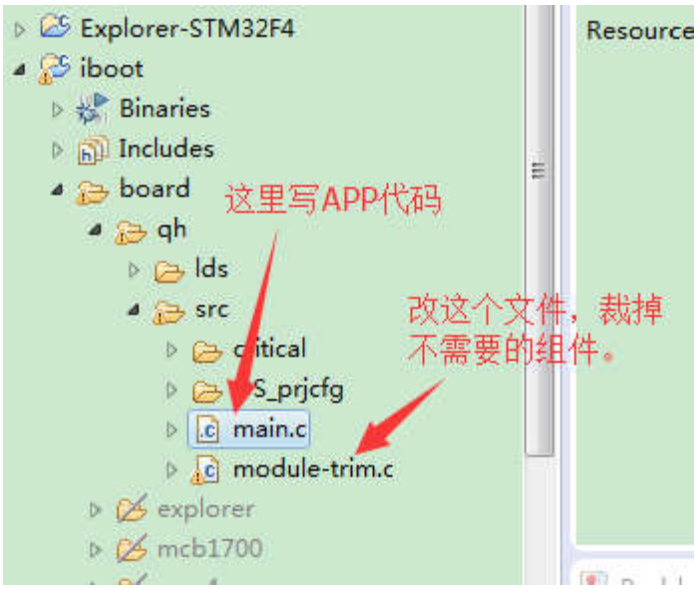

图 4-1 在 Iboot 工程中做 APP

特别提示：强烈不推荐使用无 Iboot 模式开发产品，Iboot 并不会占用多少资源，没有 Iboot，产品几乎无法升级程序。

5. Iboot 移植

本章假设你已经熟练使用 djyos 的开发工具，否则请参见文档《DJYOS 开发工具手册》。

5.1. 直接使用 sdk 包中的 Iboot

编译 Iboot 前，须先编译库，参见文档《DJYOS 库编译手册》

Djyos 的 sdk 包中，包含了一个 Iboot 工程，能够编译出所有支持的板件的 Iboot.bin 文件。用户大可不要急着制作自己的 Iboot，可先检查 sdk 中的 Iboot 有没有可以直接使用的，如有，皆大欢喜。一般来说，只要符合以下条件，即可直接使用：

1、 Mcu 型号系列相同。

2、 主频相同，晶振相同。
3、 如果有强制启动 Iboot 的硬件设计（参见 2.5 节），该部分硬件设计应该相同。
4、 如果是单片机项目，mcu 的片内 flash size 和 ram size 相同。
5、 下载代码的通信口相同。

5.2. 从 sdk 包的 Iboot 中修改

找一个最接近的板件，copy 然后修改，假设你的板件与“qh”板子最为接近，过程如下：
1、 从“djysdk\iboot\board”目录下，找“qh”目录，复制一份，并重命名为 “mybroad”。
2、 在 eclipse 中刷新 Iboot 工程。
3、 点开 Iboot 的工程配置，如下图，添加 iboot_myboard 编译项。

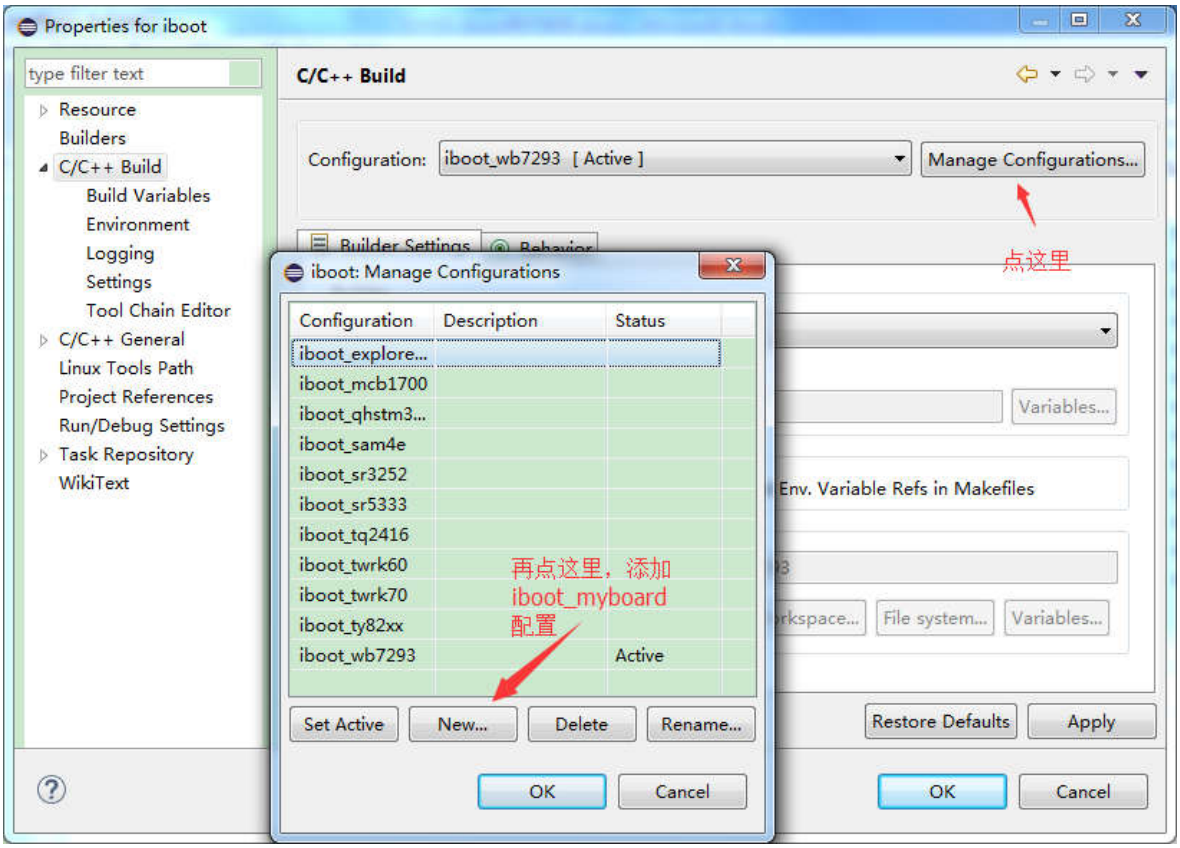

图 5-1 添加 muboard 编译项

4、 设置 iboot_myboard 编译配置中，只编译“djysdk\iboot\board\mybroad”目录。
5、 依第 5.3 节，修改“djysdk\iboot\board\myboard\src”下的 Iboot_config.c 文件。
6、 依第 5.6 节，修改“djysdk\iboot\board\myboard\lds”下的 lds 文件。
至此，工程即创建完成。

5.3. Iboot 配置

Iboot 配置就是修改板件对应的文件夹下的 Iboot_config.c。
g_IbootType 用于配置 iboot 的加载方式，枚举量_ENUM_RUN_MODE_列出了可能的值。

```
enum _ENUM_RUN_MODE_
{
    EN_LOAD_FORM_DATA=0,
    EN_DIRECT_RUN,
};
```

EN_LOAD_FROM_DATA：从数据存储器（如片外 nandflash）加载应用程序的 bin 文件。
EN_DIRECT_RUN：从程序存储器（如片内 flash）中加载应用程序的 bin 文件。
g_IbootCRC 用于配置在加载应用程序前，是否进行 CRC 数据校验。

```
enum _ENUM_USE_CRC_
{
    EN_NO_USE_CRC=0,
    EN_USE_CRC,
};
```

EN_NO_USE_CRC：加载应用程序前，无需 CRC 数据校验。
EN_USE_CRC：加载应用程序前，需 CRC 数据校验。

5.4. Ymodem 配置

Ymodem 配置就是修改与 Ymodem 数据通信软件的配置，修改该配置的文件为 ymodemcfg.c，它的存储路径是 djysdk\iboot\board\boardname\src\OS_prjcfg。

```
const char *gYmodemDevName     = NULL;       //NULL表示stdin

const u32 gYmodemBufPkgNum     = 16;         //缓存ymodem包数量

const u32 gYmodemPkgTimeOut    = 15*1000*1000; //包间隔超时时间，微秒

const u32 gYmodemTotalTimeOut = 300*1000*1000;    //ymodem 总超时时间
```

如代码所示配置项，gYmodemDevName 是配置 Ymodem 所使用的通信设备名称，如“UART1”，表示 Ymodem 接收和发送数据是通过串口设备 UART1 实现，若配置为 NULL，表明用户默认使用标准输入设备 stdin 作为串口接收和发送数据。

gYmodemBufPkgNum 配置 Ymodem 模块缓冲的数据包数，每包数据大小为 1024 字节，达到该配置包数，模块就会将数据写入到 ROM。

gYmodemPkgTimeOut 配置 Ymodem 数据包间隔的超时时间。

gYmodemTotalTimeOut 配置串口通信总超时时间。

5.5. TCPIP 配置

TCPIP 配置网络协议栈的相关配置，配置文件为 tcpipconfig.c，存放路径为 djysdk\iboot\board\boardname\src\OS_prjcfg\net。具体配置选项的说明，注释写得比较详细，此处不再赘述。

5.6. LDS 文件编写

LDS 是 gcc 的链接脚本文件，这里假设你已经能熟练阅读和修改 LDS 文件。

5.6.1. MCU 模式

Mcu 模式下，程序存储和运行都在 flash 中，flash 划分为 Iboot 和 APP 和 APP 校验信息三部分。flash 和 RAM 空间分布如图 5-2 所示。iboot 将内存块最后 16 个字节作为 BootFlg，用于标明是要启动 Iboot 或是 APP。

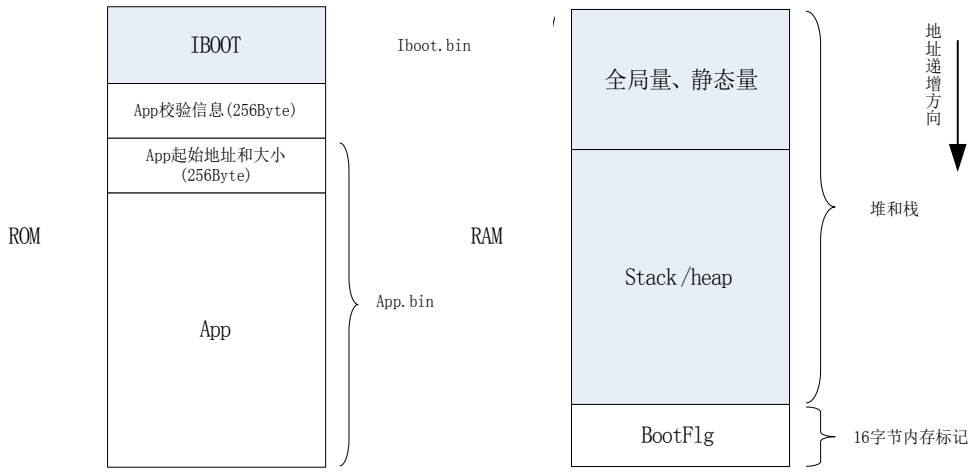

图 5-2 IAP 资源分布图

- Iboot.bin：DJYOS 的启动加载器，一般由仿真器（如 jlink）烧录到 ROM；
- APP 校验信息：debug 版本由 LDS 生成，release 版本是在应用程序升级完成后，由 IAPFS 将校验信息写入到 ROM 该位置；
- App 起始地址和大小：占用 256 字节，由 LDS 文件里面自动计算；
- BootFlg：16 字节的内存标记，标记为“RunIboot”时，则运行 Iboot，否则运行 APP。

从 IAP 资源分布图可知，IAP 模块实质上增加了对 ROM 空间的需求，而内存空间并未增加，因为 Iboot 和 APP 分时使用内存。

了解了 Mcu 模式的 LDS 资源分布后，可能大家已经开始跃跃欲试了吧，那么，接下来，以起航的 iboot 的 lds 为例，讲述 lds 文件的移植。

官方下载的 SDK，在路径 djysdk\iboot\board\qh\lds 中存放 memory.lds 和 iboot.lds 两个文件，分别定义了存储空间和资源分配。用户可直接复制 lds 文件夹到对应板件 djysdk\iboot\board\boardname 路径，稍加修改即可。

首先，对 memory.lds 文件的存储空间大小和起始地址进行修改；

其次，替换 iboot.lds 中库名称，并修改与具体板件相关文件的路径；

最后，删除或添加内容，例如，若板件没有片外 RAM，则需删除与片外 RAM 相关配置。

5.6.2. CPU 模式

CPU 模式下（例如 tq2416），程序并不在 flash 中运行，但 flash 和内存的布局却大同小异，lds 文件也只是略有不同。CPU 模式下，主内存一般用 DDR，复位会使 DDR 数据丢失，故 Iboot 和 APP 切换时，只能 reboot，不能 reset。

以 tq2416 为例，程序存储在片外 nandflash，因此在系统上电后，会将程序加载到 RAM 运行。因此，如 djysdk\iboot\board\qh\lds\memory.lds 文件中，将 RAM 空间的前 8M 用于加载程序，功能相当于 Mcu 模式中 ROM。CPU 模式的 LDS 文件修改与 Mcu 模式大致相当，此处不再赘述。

5.7. 通信方式

5.7.1. Uart

采用串口通信方式升级应用程序，所需的 ROM 和 RAM 空间小，容易实现且运行稳定，适用于小型单片机系统。djyos 实现了 YMODEM 模块，用户可通过 windows 自带的超级终端实现 IAP 在线编程。

5.7.1.1. Uart 驱动

Uart 驱动需实现 cpu_peri_uart.c，它的存放路径建议在 bsp/cpudrv/cpuname/src。

YMODEM 通信协议访问串口使用标准的 driver 设备驱动模式接口，串口标准驱动请参看《都江堰操作系统用户手册.docx》第 12 章，即设备驱动模型。在章节 12.4 中，详细介绍了串口驱动模型的使用方法与收发数据过程，用户可参考官方下载的 SDK 起航板件的串口驱动程序实现串口驱动代码编写，存放路径是 bsp/cpudrv/stm32f1xx/src/cpu_peri_uart.c。

5.7.1.2. Uart 接口调用

下面以起航板件为例，简述 iboot 工程通过 Uart 实现 IAP 功能的接口调用方法。

源码存储路径是 SDK 的 djysdk\iboot\board\qh\src\module-trim.c。

```
void Sys_ModuleInit(void)
{
    ...
    Stdio_KnlInOutInit( 0 );                          //1
    ModuleInstall_Sh(0);                              //2
    ModuleInstall_Driver(0);                         //3
    ModuleInstall_Multiplex(0);                       //4

    ModuleInstall_FileSystem();                       //5
    ModuleInstall_BootFS();

    ModuleInstall_UART(CN_UART1);                    //6
    Driver_CtrlDevice(ToDev(stdin),CN_UART_START,0,0);
    Driver_CtrlDevice(ToDev(stdin),CN_UART_SET_BAUD,115200,0);

    OpenStdin(gc_pCfgStdinName);                     //7
    OpenStdout(gc_pCfgStdoutName);
    OpenStderr(gc_pCfgStderrName);

    ModuleInstall_Ymodem(0);                                  //8
    Ymodem_PathSet("/iboot");
    ModuleInstall_IAP();                                        //9
    ...
}
```

启航板 iboot 串口使用 IAP 功能接口调用说明如下：

如//8 所示，启航板装载 Ymodem 时，使用的参数是 NULL，代表 Ymodem 从 stdin 读写数据，即标准输出输入的配置名称 gc_pCfgStdinName 为“/dev/UART1”，因此，需//6 初始化串口设备 CN_UART1，需//2 加载 Driver 模块，需//1 和//7 标准输入输出模块。

起航板件通过 windows 超级终端升级应用程序，因此需要 shell 模块接收“download”升级命令，因此需//2 装载 shell 模块。

Ymodem 接收完数据，通过文件系统写入到应用程序指定位置，因此需//5 文件系统初始化，需//8 设置文件路径。

最后，装载 IAP 模块。

5.7.2. 网络

使用网络 FTP 工具升级的最大优点就是速度快，使用方便。DJYOS 已经实现了 ftp 协议，用户需配置网络相关的选项及实现网络驱动程序。

对于网络驱动程序，《都江堰操作系统用户手册.docx》21.5 章节中介绍了网络驱动程序的相关

介绍，用户可参考官方网络下载的 SDK 例程，实现网络驱动程序。如 ATMELSAMV70 的网络驱动程序路径为 bsp/cpudrv/AtmelSamV7/src/cpu_peri_gmac。

网络模块的初始化及相关的配置选项说明，以 ATMELSAMV70 为例，配置文件存放的路径为 djysdk\iboot\board\wb7293\src\OS_prjcfg\net，其中，boardnetcfg.c 中调用网络驱动程序初始化和 tcpip 模块装载，tcpipconfig.c 中是 tcpip 配置的相关常量，用户通过修改配置常量值，配置 tcpip。

5.7.3. 其他

除串口外，YMODEM 可通过 SPI、IIC、USB 等通信方式升级应用程序。当然，用户需将 SPI 或 IIC 器件抽象成设备（同串口设备），调用 Driver_DeviceCreate，创建设备，并将设备名称通过字符串指针参数传递到 ModuleInstall_Ymodem。

调用 Driver_DeviceCreate 创建设备时，读写和控制函数接口标准化为 fnDevWrite、fnDevRead 和 fnDevCtrl，用户可将 SPI（IIC 或其他）通信方式的读写及控制函数包装，实现设备对象的创建。

有关设备驱动模型，请参阅《都江堰操作系统用户手册.docx》第 12 章，设备接口函数原型请参阅 SDK 路径 djysdk\djysrc\component\include 下面 driver.h 文件。

5.8. IAPFS 驱动

通信程序收到应用程序 bin 文件后，调用标准文件系统接口把 bin 文件写入 flash，如果是标准文件系统，这里就不多说了。

如果数据直接写入 flash，DJYOS 提供 IAPFS 文件系统，把应用程序 bin 文件写入到 flash 对应的地址，对上提供文件系统接口，用于应用程序的在线升级。

这样，无论哪种存储模式，FTP 和 ymodem 都用统一的方法写入 bin 文件。

IAPFS 完成如下功能：

- 提供标准的文件操作接口，ymodem 模块和 ftp 模块调用标准接口操作文件；
- 调用底层硬件驱动，实现对 ROM 的读、写、擦除操作；
- 写入图 5-2 所示的 AppCtrlRom 校验数据。

对于 IAP 的编程人员，须根据具体的 ROM（一般是 FLASH）特点，实现 IAPFS 的驱动，实质上是完成对 FLASH 操作的读、写、擦除等操作。

5.9. 专用硬件

如第 2.5 节所述，用户都需要通过 IAP_RegisterForceBoot()实现专用硬件注册，如果没有专用硬件，则无须注册即可。其函数原形为

```
typedef bool_t (*IAP_ForceIbootFunc)(void);

void IAP_RegisterForceBoot(IAP_ForceIbootFunc ForceBoot);
```

6. APP 开发调试

本章假设你已经把 Iboot 烧录到目标机了，否则，参见第 2.1 节。

详细的仿真调试设置，请参见文档《DJYOS 开发工具手册》。

如果是 debug 版本，可以直接在仿真环境下载代码和调试。

Release 版本则必须用 Iboot 下载，才能用仿真器调试。

Iboot 引入检查 APP 是否完整正确的功能后，对仿真调试是有影响的。回忆一下 3 章所讲的 APP 加载过程，是不是要检查 APP 的完整正确性标志？这些标志是 IAP 写入的，用仿真器或者其他 flash

烧写工具是否会写入这些数据呢？暂时不会，等 djysdk 完善后，就会了。

这么说，只要不是用 IAP 下载的代码，根本不会执行你要调试的 APP，调试也无从谈起了。每次修改代码，都要用 IAP 下载，烦死你不偿命。

我们考虑到这个问题，对于 debug 版本，在本该出现“完整正确性”标志的地方，我们放上“dbg”标志，这个工作由 debug.lds 完成。Iboot 检测到这个标志，就跳过检查代码的完整正确性，使之能够正常运行。“dbg”串会包含在 ELF 文件中，仿真器加载时会被烧录到 flash，故能正常运行到 APP。

release 作为 APP 调试完成后的发布版本，一般不需要调试。很多产品会采用 O2 级别优化，可调试性也很差。但不可能 100%避免调试，尤其查找现场发现的问题。只是，release 版本代码必须使用 Iboot 下载才能调试，调试方法跟 debug 版本是一致的。

7. API 接口函数

7.1. ModuleInstall_IAP：IAP 模块加载

ptu32_t ModuleInstall_IAP(void)

参数：

无。

返回值：

0。

说明：

Iboot 和 APP（包括 debug 和 release）都须加载 IAP 模块，建议在 module-trim.c 中的 Sys_ModuleInit 中调用本函数。

7.2. IAP_GetAPPStartAddr：获取 APP 起始地址

u32 IAP_GetAPPStartAddr(void)

参数：

无。

返回值：

APP 运行时起始地址。

说明：

7.3. IAP_GetAPPSize：获取 APP 大小

u32 IAP_GetAPPSize(void)

参数：

无。

返回值：

APP 运行代码尺寸，单位字节。

说明：

7.4. IAP_GetAPPCRC：获取 APP 的 CRC 值

u32 IAP_GetAPPCRC(void)

参数：

无。

返回值：

获取 APP 的 CRC 值。

说明：

此处获取的 CRC 值是由 IAPFS 对下载的应用程序 bin 文件计算得出。

7.5. IAP_GetAPPStartCodeRamAddr：获取加载到 RAM 代码起始地址

u32 IAP_GetAPPStartCodeRamAddr(void)

参数：

无。

返回值：

应用程序代码加载到 RAM 的起始地址。

说明：

对运行在 ROM 中的应用程序，此函数获取的数据并无意义。

7.6. IAP_GetAPPCodeRamSize：获取加载到 RAM 代码尺寸

u32 IAP_GetAPPCodeRamSize(void)

参数：

无。

返回值：

应用程序代码加载到 RAM 的大小，单位字节。

说明：

对运行在 ROM 中的应用程序，此函数获取的数据并无意义。

7.7. IAP_LoadAPPFromFile：从文件加载 APP

bool_t IAP_LoadAPPFromFile(void)

参数：

无。

返回值：

若返回，必然为 false，否则将跳转到 APP 中运行。

说明：

该函数适用于 CPU 模式，当 iboot 完全启动后，调用该函数，从文件系统加载应用程序，若应用程序加载正确并校验通过，则直接进入 APP 中运行，否则返回为 false。

8. 附录一：超级终端安装

解压该压缩文档，并按照 readme.txt 操作，即可完成超级终端安装。本安装过程只征对 windows 7 操作系统，windows XP 自带超级超级终端，因此无需安装。

hypertrm.zip

9. 附录二：FTP 安装

解压该压缩文档，并双击 flashfxp.exe 便可运行 FTP 程序。

FlashFXP_CN.rar