si 版本中添加 driver

djyos 的 si 版本中，driver 和操作系统是一起编译到一个可执行映像中的，本文档主要有两部分内容：

1、 如何添加、修改 driver 代码。

2、 编译和烧录 driver。

本文档是一个 Quick start，详细的 driver 代码如何编写，参看《都江堰操作系统与嵌入式系统设计》一书。

1. 创建 driver 目录

1.1. si 版本的目录结构

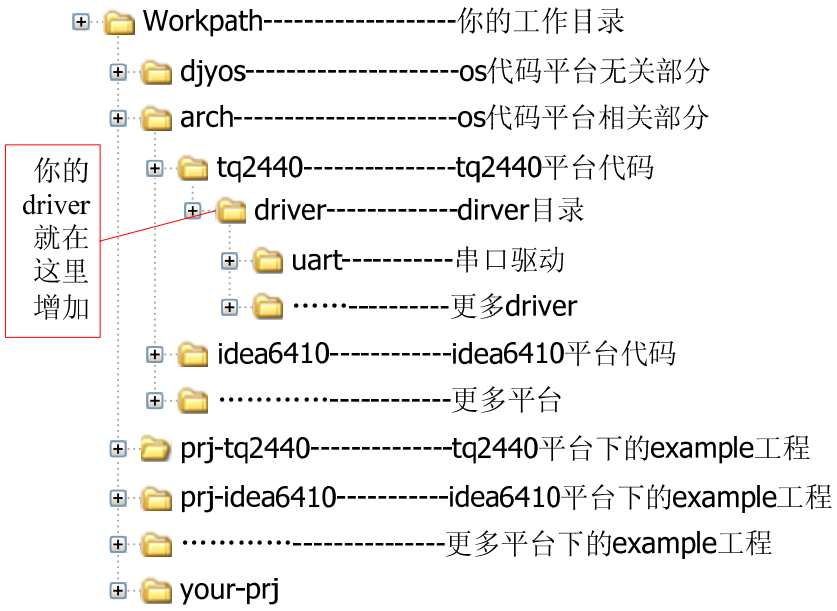

图 1 si 版本目录结构

这是 djyos 整体目录结构，arch 目录包含了多个移植平台的代码，每个平台一个子目录，例如 tq2440 子目录下存放的是 tq2440 开发板的移植代码。在 tq2440 平台下的所有 driver 都在 driver 子目录下。增加 driver 就从 dirver 目录下创建你的 driver 目录开始。

1.2. 创建 driver 目录

假定你要创建一个名为“my-driver”的驱动程序目录，你需要经过以下步骤：

1、 复制 driver 目录下任意一个目录，并把它改名为“my-driver”。

2、 修改 driver 目录下的 makefile 文件，把 my-driver 添加到这一行中：

subdir =uart key nude_io flash_chip

改为：

subdir =uart key nude_io flash_chip my-driver

3、 修改 my-driver\makefile 文件，把这一行：

sysload =uart.c

中的 uart.c 改为你的驱动程序源文件，如果有多个源文件，则列在一起，用空格隔开（不

是逗号），汇编代码也一样，如下：

sysload =src1.c src2.c src3.s

4、 如果 my-driver 下有包含源程序文件的子目录，子目录名为 fold1，则把该子目录添加到 subdir 变量中，有多个子目录则空格隔开，如下：

subdir = fold1 fold2

2. 编译工程

特别提示：si版本是应用程序和操作系统编译在一起的，图 1中的工程目录必须存在，即使djy_main函数是一个空函数也必须存在。

本文档假设你已经按照《建立 windows 下 djyos for arm 的编译和调试环境.doc》文件中的方法安装好 gcc 编译工具。

在 windows 下，点“开始-运行”，在运行对话框：

中输入 cmd，点“确定”进入命令行方式，然后进入工程目录（主控 makefile 文件也在这个目录中），**注意，不是进入安装 cygwin 时带的“Cygwin Bash Shell”环境**。在这个目录中执行 make 命令即可编译。假定工程目录是 g:\djyos\si\my-prj，可用下列 4 条命令分别编译不同目标文件：

g:\djyos\si\my-prj>make clean　　删除所有编译结构文件。

g:\djyos\si\my-prj>make debug　　编译产生 debug.elf 文件，未经编译优化，并且包含大量调试信息，可用于仿真调试。

g:\djyos\si\my-prj>make run_inram　编译产生 run_inram.bin，编译器设为 2 级优化，该文件可烧录到 flash 中，运行时自动 copy 到 ram 中执行。

g:\djyos\si\my-prj>make run_inflash　编译产生 run_inflash.bin，编译器设为 2 级优化，该文件可烧录到 nor、flash 中，直接在 norflash 中运行，适合像 cm3 这样的单片机。

编译产生的结果文件：

*.elf：elf 格式的可执行文件，包含符号表，还可包含更多的调试信息。

*.bin：可直接烧录的二进制可执行文件。

2.1. 何时应该使用 make clean 命令

make 有一个特点，就是如果被依赖的文件比目标文件更新的话，就会重新编译目标文件，否则不会重编译。比如两次编译间，a.c 文件没有被编辑过，那么被依赖的文件 a.c 就不比目标文件 a.o 新，因此第二次编译时 a.o 就不会被重新编译，但有时候，我们不希望这种“查新”机制起作用，就要用 make clean 命令把目标文件删掉，强制重新编译。

1、 修改头文件

原理上讲，如果依赖关系做得好，头文件也成为.o 文件的依赖文件的话，头文件修改后，编译

器能正确重编译依赖该头文件的目标文件，但事与愿违，有些时候头文件与 c 源文件的依赖关系很复杂，并且在编程过程中会有调整，要完全描述这种依赖关系变得很困难，且 makefile 会被弄得很复杂难读。因此，djyos 的 makefile 不描述头文件和 C 文件的依赖关系，故在修改头文件后，需要执行 make clean 命令删掉所有.o 文件，强迫重新生成所有.o 文件。

2、 目标改变

比如原来执行过：

make debug 编译调试版本

现在要执行

make run_inram 编译可直接执行版本，由于在执行 debug 时，已经产生了.o 文件，而.c 文件又没有修改，故不会重新编译，而是直接把编译 debug 时产生的.o 文件重新连接。而 run_inram 和 debug 的命令行参数并不一样，该.o 文件是按 debug 的命令行参数编译的，不适合 run_inram 使用，必须重新按 run_inram 的参数产生所有.o 文件，故需要在执行 make run_inram 之前用 make clean 删掉所有.o 文件。

3. driver 编程

前面把工程架子都搭好了，但还没有编写一行代码呢，driver 的功能，归根究底还是要通过代码才能实现。前面都是搭台子吆喝，光吆喝是不行的，练把式还得动真格，沉住气，真功夫马上登场。虽说是真功夫，但这里还是只讲招式，内功秘笈（driver 原理）还得看《都江堰操作系统与嵌入式系统设计》。

3.1. driver 模块初始化

driver 模块应该有一个初始化函数，该初始化函数的命名方式必须以“module_init_”开头，例如 key 模块的初始化函数是 module_init_keyboard，函数原型必须是：

bool_t module_init_ keyboard (void)

为什么要规定函数名，以及函数原型为什么没有参数，这跟支持项目经理组织项目团队有关，其功效主要在 dlsp 和 mp 版本中体现，这里先卖个关子，等 dlsp 出来后再统一说明。si 版本中遵守这个规定，可以使你的 driver 能够兼容未来的 dlsp 版本。

初始化函数定义好后，一般在 appinit.c 文件中的__djy_main 函数中调用，但这不是固定的，你可以在应用程序的任何地方调用。

module_init_xxx 函数的功能是：

1、 执行 driver 模块私有的初始化工作，不要误解为硬件初始化工作，djyos 用的是“泛设备”概念，driver 驱动的不一定是硬件，还可能是系统中的其他软件模块。

2、 调用 dev_add_root_device 函数或 dev_add_device 函数把新设备添加到系统设备队列中。

3.2. 实现 driver 功能

实现 driver 的功能，即实现泛设备的两套接口：读、写、控制，左右手各一套，这两套接口的详细说明参看《都江堰操作系统与嵌入式系统设计》一书，以及参考共享代码中的 uart.c 和 key.c，本文档是 Quick start 文档，不做详细说明。