DIDE设计大纲

编写：罗侍田

# 源码管理

## 板件分类管理

存在的问题：上传代码到git时，需要手工删除公司板件，且这些板件“藏”在不同的地方，比较繁琐；同样，如果允许外部用户上传，也存在同样的问题。改为：

1. bsp/boarddrv/demo目录下，放置官方提供的板件，并且是测试过没问题的，这些板件在DIDE板件选择中列出。每块板下面，增加startup子目录，例如bsp/boarddrv/demo/apolle-stm32f7/startup目录。
2. 增加bsp/boarddrv/ staged目录，把曾经用过，暂时不打算维护的板子放进去。本目录板件不在DIDE中列出，如果需要用，则测试后，copy到demo目录。
3. 原bsp/startup目录删掉，startup目录下跟板走的代码，移到boarddrv目录下；原来跟CPU走的代码和头文件，转移到cpudrv目录下。
4. 用户自己的板件，放在bsp/boarddrv/user目录，现在发布时很被动，从SVN上下代码后，总是要手工一点点把公司代码删除。

## 内部SVN和外部git管理

开发用内部SVN服务器，按当前方式维护。

开源共享使用git，git中分为开发主线master和发布tag。

开发主线经常更新，尽量跟trunk同步。

发布主线，每当要发布一个稳定版本，就与trunk同步。

tag：每次发布主线更新前，打个标签。

DIDE的发布，内置tag发布主线地址。

# 资源描述

## 组件描述

用于描述具体组件：

1. 所有组件均由cfg文件描述，包括功能组件、芯片驱动组件、CPU片内外设驱动组件等，cfg文件放在组件相应的源码目录，文件名格式：“compont\_name.cfg”。
2. 组件名，该名字用于显示在界面上。可重名，源码目录不一样就行
3. 组件属性：第三方组件、自有组件、必选组件。如果是第三方组件，会为该组件单独编译一个.a文件。列出必选组件的原因，是可能需要在组件配置界面上配置该组件的参数。
4. 修改源码目录结构，使一个目录只保存一个组件，但一个c文件内包含多个组件的除外，例如cpu\_peri\_uart.c包含多个uart端口。
5. DIDE生成工程时，把该文件copy到工程目录，并改为.h文件。
6. cfg的文件内容之一：组件初始化代码，也在cfg文件里面，（可能是空的，代表该组件不需要初始化直接使用），DIDE用来生成initprj.c。
7. cfg的文件内容之二：组件参数，DIDE用来生成compontname\_config.h放到工程目录中
   1. 初始化函数的参数
   2. 条件编译用的常量，或控制资源用的常量。
   3. include “compontname\_config.h”时，应加注释：//本文件由配置工具生成，在工程文件夹中
8. cfg的文件内容之三：依赖关系描述
   1. 本组件是否必须首先初始化，例如原始IO初始化函数
   2. 列出全部父组件名，可以有多个。
   3. 列出全部先导组件名，即必须先于本组件初始化的组件，可以有多个。
   4. 列出全部互斥组件名，即与该组件不能同时存在的组件
   5. 本组件是否必须最后初始化，例如动态内存初始化函数。
   6. 本组件是否必须在系统启动后初始化（即main函数里面）。

问题：

如何维护依赖关系？一个被许多组件依赖的组件改名后，其他组件的描述都要修改，否则就会乱掉。改善方案：定期检查源码中cfg文件的内容，如果出现没有与之匹配的被依赖组件的情况，告警。

### 组件描述改进

存在问题：原方案导致目录太多，文件太多，很多时候，是一个文件即一个组件，要加上一个xml文件和一个h文件，太太太多了。且每个有配置的组件，必须用IDE配置才能编译，否则被include的文件就会不存在。

改进要求：不能大幅增加文件；不需要完成配置，直接可以编译；不影响可读性；每个组件独立完成配置，无需要手工修改的公共配置文件。

方案一：

C文件自描述和独立描述相结合，不再使用xml文件。

独立描述指使用单个文件component\_config\_name.h描述整个组件，适合于大型组件，特别是第三方组件，这种组件由一个目录下的许多文件和子目录组成。

自描述指的是在每个c文件开头的地方，用特定的注释格式描述和配置该组件，适合由很少文件构成的组件。

如果某目录中有component\_config\_name.h文件，则优先使用，具体的c文件，不再自描述。

1. .c文件中有默认的宏，如果配置文件中没有提供相应宏，则使用默认配置，并在编译时提供告警。
2. IDE只生成一个h文件，只需要include一个文件即可，即使该文件是个空文件，也不影响阅读代码以及编译。
3. 第三方组件的配置方法，依照其原来的配置方法，component\_config\_name.h文件只描述组件名、依赖关系以及初始化代码等必须的，不做宏配置。（后续会为每一个第三方组件做专用的配置插件）

方案二：（不取）

改进当前xml，一个xml文件可以描述多个组件，每个组件列出该组件包含的c文件。仍然每个xml文件对应一个h文件用于配置组件。问题：文件减少量不显著，且xml、h文件和IDE产生的h文件怎么命名？

### 图形化编辑组件描述信息

eclipse中，编辑器界面上，增加一个“CFG”按钮，按下后，弹出界面，用图形化的方式，编辑配置的内容，然后生成配置文本，插入光标所在位置。

以“//@#$%component configure”和“//@#$%component end configure”作为配置代码标识。

如果文件中尚无配置标识，按下“CFG”按钮，则用初始值填充编辑界面。

如果文件中已经有配置，则读取并填充到编辑界面，以供修改。

## CPU描述

### CPU架构与家族目录和文件

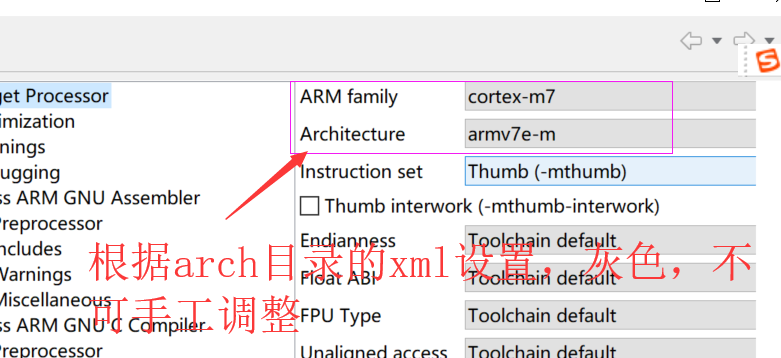
从系列——具体arch，采用逐级描述模式，系列的描述文件名为arch-serial.xml，arch的描述文件名为arch.xml。

xml文件的内容，要反映出编译器中与CPU相关的选项，例如-march，-mcpu等，还要指出工具链名称，一般是在第一级的arch-serial.xml中。

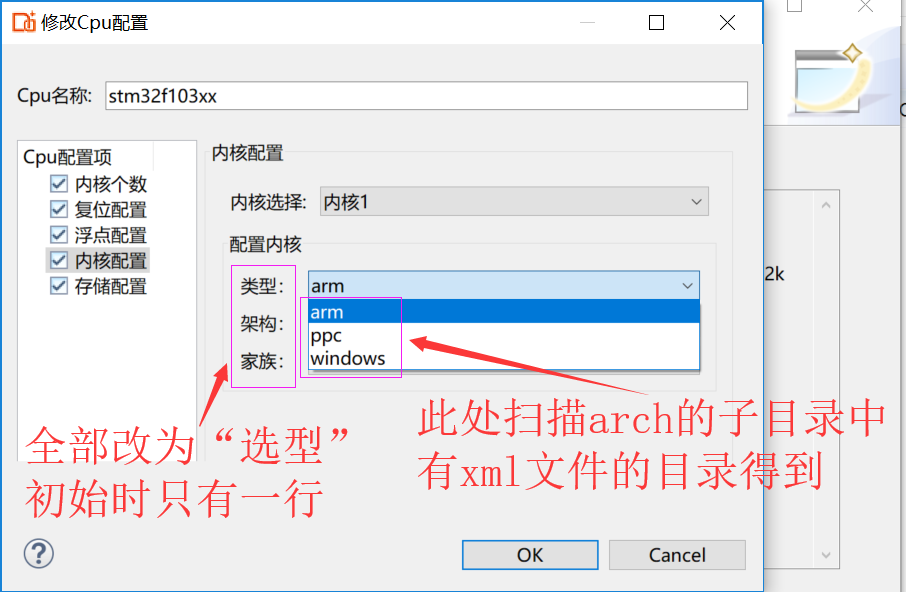
扫描时，直到arch.xml为止，不再扫描子目录。任何路径，除非不出现arch-serial.xml，只要出现，则最后必有一个arch.xml文件，否则报错。

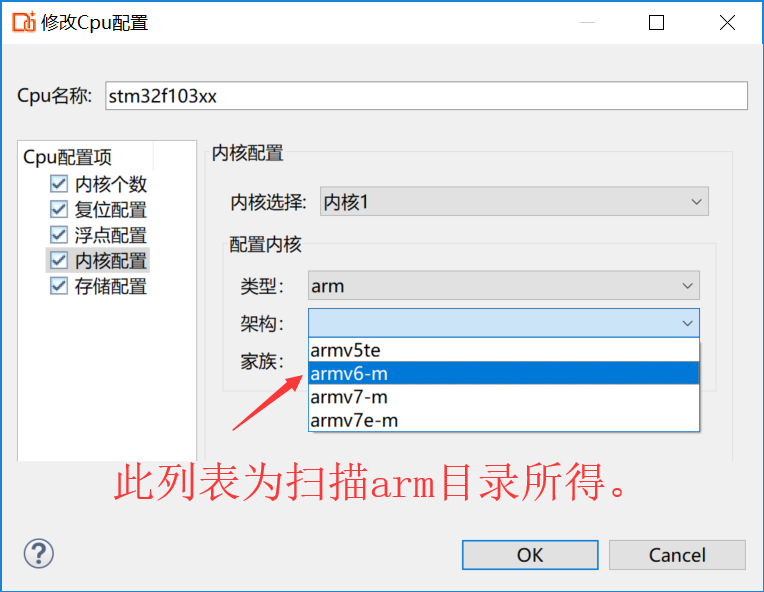
添加编译文件的方法，与cpudrv目录相同。

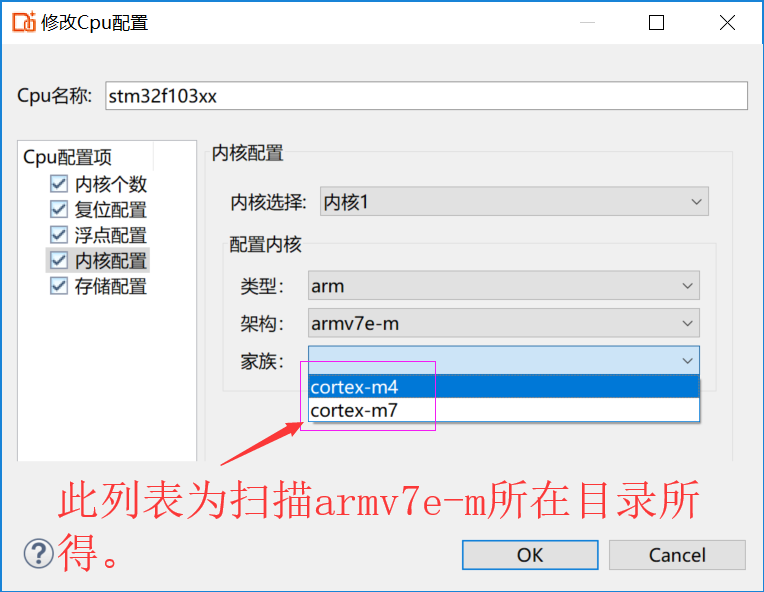
工程配置界面：



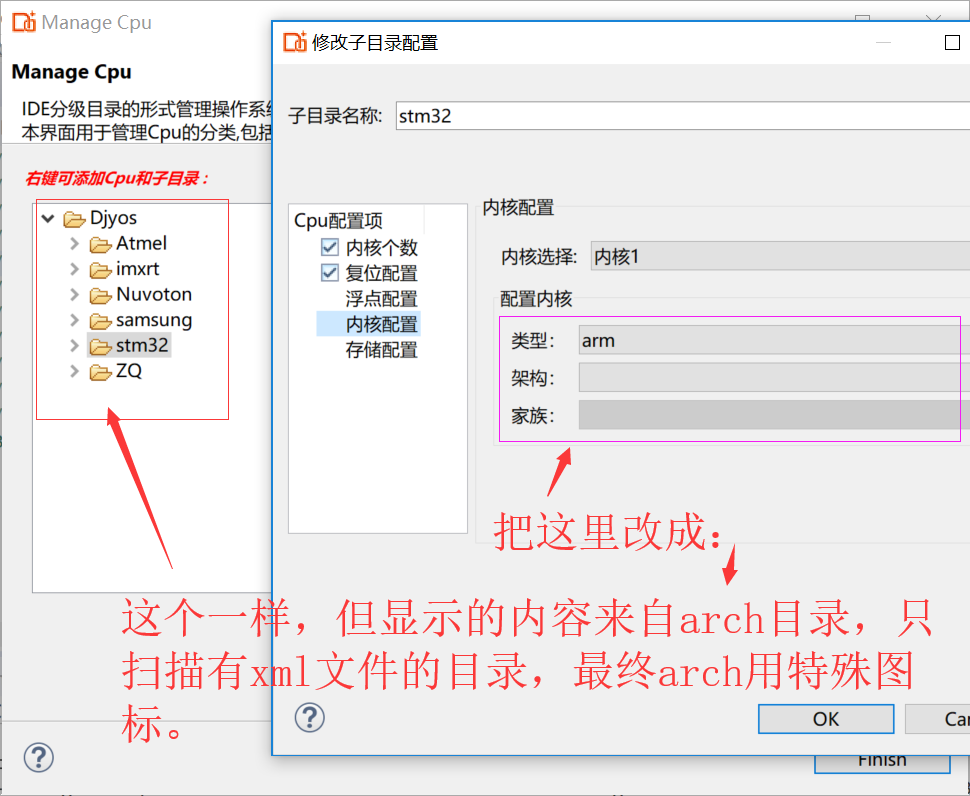
CPU配置界面方案一：







方案二：



### CPU芯片驱动目录和文件

与创建工程并列，增加“new CPU”菜单项，用于添加CPU。

CPU描述文件放在cpudrv路径下相应的CPU目录中，现在许多CPU都是家族化的，同一厂家的同一家族有许多细分型号，同一家族CPU大部分相同，细分型号有些许不同。

cpudrv目录下，文件夹也按照家族分级创建，每一级目录，都可以放置一个cfg文件，用于描述相应家族的公共属性，文件命名为“cpu\_group\_ name. cfg”。到具体芯片型号目录，放置一个文件名为“cpu\_cpuname. cfg”，cpuname即厂家命名的CPU型号，不理会只跟封装和温度范围等与软件无关的部分后缀。命名示例：

有目录“cpudrv/stm32/stm32f7/stm32f767/”，则：

在“cpudrv/stm32/”下，有文件cpu\_ group\_ stm32. cfg文件，描写所有stm32的公共特征。

在“cpudrv/stm32/ stm32f7/” ，有文件cpu\_ group\_ stm32f7. cfg文件，描写所有stm32f7的公共特征。

在“cpudrv/stm32/ stm32f7/stm32f767/” ，有文件cpu\_ stm32f767. cfg文件，描写所有stm32f767的公共特征。注意文件名中没有“group”。

除了列举部分公共的属性外，还要能在界面中添加新属性。

包含以下内容：

1. CPU核数量
2. 多核公共能访问的存储器（rom、flash、ram）地址及容量
3. 每个CPU核的内核和架构（每个核都要单独描述，因为有异构的）
4. 每个CPU核私有的存储器（一般是ram）的地址和容量
5. 其他需要的内容。

## 板件描述

与创建工程并列，增加 “new board”菜单项，用于添加板件。

创建新板件时，如果要创建新CPU的，调用“new CPU”菜单命令。

板件描述放在板件目录下，以“board\_”开头的xml文件，描述内容：

1. 板件名称，一般与目录名相同。
2. 主时钟晶振频率、rtc时钟频率（可以无）
3. 板载存储器（rom、flash、ram）地址及容量，不含CPU内部的。
4. CPU名字，必须与“CPU描述”节中的“cpu\_cpuname.xml”对应。如果是多CPU板件，把所有CPU都列出来。
5. 按每个CPU：
   1. 列出该板件实际引出的片内外设组件名，必须是该CPU目录下的组件名。
   2. 列出该CPU连接的chip 驱动组件名。

## 扫描板件和CPU信息

当CPU和板件选定后，依以下步骤可得到创建工程所需的硬件信息。

1. 扫描cpudrv目录，在名字为“cpu\_cpuname.xml”的文件中，找到所选择的CPU对应的xml文件。
2. 从该xml文件沿路径往回找，直到cpudrv目录，找到全部名为“cpu\_ group\_xxx.xml”的文件，合并所有xml文件的内容。
3. 从合并的xml文件中，可以得到该CPU的描述。
4. 扫描cpudrv目录的过程中，可以得到该CPU的外设驱动组件列表。这样做的好处是，只扫描我们已经提供驱动的外设。
5. 分析板件描述文件“board\_xxxx.xml”，得到该板件支持CPU外设驱动组件和chip驱动组件，以及板载存储器描述。
6. 校验这些组件是否存在相应的驱动程序，存在则把这些组件添加到编译序列中。

# 工程配置

## 创建工程

创建工程的步骤：

1. 新建或选择板件和CPU。
2. 按上一节的方法，扫描板件和CPU相关源码和相关xml文件。
3. 设置主频及其他需要设置的参数。
4. 打开内存设置界面，配置内存（用于生成lds）。
5. 勾选该工程需要的组件，并配置。
6. 生成工程文件，除cproject和project文件外，还应包含：
   1. 组件配置xml文件。
   2. lds文件
   3. 生成initprj.c文件，不再使用module-trim.c文件。
7. 把需要编译的文件的链接添加到工程中，方法如下：
   1. 扫描CPU时，找到CPU目录，把该目录的全部代码加到工程中。并把祖先目录中的代码也加到工程中。
   2. 扫描boarddrv目录和chipdrv目录时，照此办理。
   3. arch目录亦如此处理。

## 组件配置

由于板件和CPU已经选定，就可以用2.4节所示的方法，扫描源码目录，得到该工程允许勾选的硬件相关组件。再加上第三方组件和操作系统自有组件，就可以生成组件配置界面了。

勾选和配置参数后，按扫描结果，生成initprj.c文件和各模块参数的头文件。

任何时候，点工程的property，可以修改配置。

## 设计连接脚本

这是用于自动生成lds的功能。

在工程的property设置中，提供“设计连接脚本”菜单。

“设计连接脚本”界面通过读取并解析targetname.lds文件生成，创建工程时不进入此界面。

### 默认配置

大多数用户，是搞不清楚目标文件和段的，DIDE应提供默认的lds文件。放在boarddrv/boardname目录下（release和debug，APP和Iboot分别一个，共四个），新工程直接copy该lds文件到工程目录。property上有“恢复默认值”按钮，也是copy该lds文件的内容。

### 固定配置

针对具体板件，有些段是必须固定地址的，例如initcpu.o，需要在boarddrv/boardname目录下放置一个文件说明，该文件格式，详细设计中设计。DIDE在内存配置界面中，固定配置灰色显示，不可修改。

### 存储区定义

用于生成memory.lds文件。

注意，创建新板件时，描述的是该板件的物理上的内存配置，这里讲的是flash和RAM如何划分。

存储器被划分为多个区，描述每个区的名字、地址、长度、属性。

Iboot工程：独立描述。

Iboot+APP工程：两者共享存储器描述。

APP工程：须指定相应的Iboot工程，或者指定Ibootsize。

示例：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Address | Size | Property | 空闲区间建堆名字 |
| InnerFlash |  |  | RX |  |
| RAM1 |  |  | RXW | sys |
| nocacheram |  |  | RW | extram |
| extram |  |  | RW | extram |

### 输入段列表

列出已知的段，这些段可以拖动到内存表中。

段包含以下内容：

1. 扫描用户工程中所有.o文件，提取用“**\_attribute\_\_((section("xxx")))**”命名的段。在界面中提示用户先编译。扫描.c乎？
2. 操作系统库中已知的段，在boarddrv/boardname目录下描述
3. gcc编译产生的标准段，例如(.data)
4. 导入工程文件目录，可以看到所有.o文件。

### 存储定位表

指定具体的段放在什么地方。

DIDE列出该工程的所有文件和段，工程中的好办，库中的呢？

不编译，就不知道有什么库啊？故需要先编译一下。然后扫描操作系统库以及APP中的全部.o文件，从elf格式中找出全部有加载属性的段。

剩下一个问题，keep的目标代码在库中的问题，解决方案有2个，暂用方案2：

1. 改gcc，修改ld过程。
2. 生成编译命令时，先扫描lds文件，把带keep的文件列入编译命令。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 所属文件 | 输入段名 | 加载时存储区 | 运行时存储区 | 是否keep | 位置 |
| 可列出多个文件。可用通配符 | 从输入段中选择，或者手工编辑 | 从存储区定义中选择 | 从存储区定义中选择 |  | 运行区的：“首、尾、中”位置 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 应用框架

应用框架可以指一类应用场景，例如图形应用框架、工业串行总线框架等。框架配置是比组件配置更高一个层次的配置，选中一个应用框架，相当于选中了与该框架相关的若干组件，并进行集中配置。DJYOS提供若干框架，用户可以按需选择0个或多个，DIDE将按照用户配置的结果，生成相应的代码。按框架配置，用户可以不深入了解其应用所涉及到的技术细节，更加方便用户使用，节省用户学习时间和成本，更是降低了配置错误导致bugs的风险，使产品更加稳定可靠。自动生成框架所需的代码，也节省了用户开发时间。

创建工程时，选择板件之后，将进入应用框架配置界面，可以勾选需要的框架，并进行配置。例如，选择了“工业串行总线框架”，你需要配置本机是主机还是从机，或者既是（上级网络）的从机，又是（下级网络）的主机，需要配置通信设备文件的名称，通信口baud等参数。

工程属性界面中，也可以显示该工程选择了哪些框架，并可以修改配置，但是，不能增减所选择的框架。

在djysrc目录下，增加profile目录，该目录下，放置一系列xxx.prf文件，每个文件用于配置一个框架，每个文件应该包含如下内容（文件格式由详细设计确定）：

1. 框架名，该名字将出现在配置界面的选择列表中。
2. 框架初始化代码（可能是空的，代表该框架不需要初始化直接使用），DIDE用来生成initprj.c。
3. 框架包含的组件，用组件名表示，组件名定义参照第2.1节。
4. 框架参数配置，DIDE用来生成profile\_name\_cfg.h放到工程目录中
   1. 初始化函数的参数
   2. 条件编译用的常量，或控制资源用的常量。
   3. 框架需要引用的设备或文件，一律使用文件名。
   4. include “profile\_name\_cfg.h”时，应：//本文件由配置工具生成，在工程文件夹中

用拖放方式而不是勾选方式选择框架，因为有些组件可能会部署多个，则拖多个过去。

# 调试方案

## CPU运行记录

记录CPU运行过程需要目标机配合，在配置工程时，须打开此功能。能观测多长时间，取决于切换频繁度以及配置的buf空间。

### 数据记录

记录CPU每时每刻都在干什么，记录中断号、事件号、事件类型、类型名称等信息。时间是CPU开始运行以来的64位时间。

如果是MP版本，还要记录事件所属进程。

如果是AMP或SMP版本，则要分别显示所有CPU的运行时间轴。

注：忽略实时中断。

记录方法：记录在内存中，循环记录，记录满以后，新记录覆盖老记录。记录使用的内存量可在IDE中配置，配置为0则没有这个组件。

在事件切离时记录，中断引擎中进入中断和离开中断时记录，包括tick中断，结果是，记录的数量会非常多。

### 调取数据

调取数据分为在线调取和离线调取。

在线调取即在debug环境（含release版本）下，调试中断中响应相关命令读取buf中的数据，上送给调试器。IDE每200mS发送一次读取数据的命令。在线调取的数据，只要计算机内存够，就一直保存。为此，IDE需要知道系统有多少可用内存，不能耗光。

离线调取主要用于现场调试，设备运行过程中，用网络ftp、串口ymoden等方式，把数据作为文件调取上来，供离线分析工具分析。需文件系统支持，为CPU运行记录buf设计一个只读内存文件系统，一同被调取的还有其他调试数据，比如内存消耗情况、各事件占用的时间统计等。

每次调取数据后，立即清空。

### 显示

调试界面可显示CPU实时运行情况或离线数据。

在CPU运行图窗口上，有“实时”、“离线”、“启动”、“停止”按钮。

“实时”则象示波器那样，实时刷新，此时“启动”和“停止”按钮是可用的。

离线则需要打开文件，此时“启动”、“停止”按钮是灰色的。

在时间轴上，显示CPU运行图，时间单位是uS，以不同的颜色显示中断、事件运行时间，须显示中断号、事件号、事件类型号、事件类型名。

可单独观察某一事件、某一中断的运行时间，也可以分类分组观察。

能像示波器那样放大、缩小视图。

## 核心数据显示

### 事件和事件类型控制块显示

用于显示当前事件控制块的使用情况。

1. 用图形方式，显示全部事件控制块和事件类型控制块的当前状态。
2. 用不同颜色显示事件控制块的运行状态，以区分“运行中”、“尚未分配线程”、“延时中”、“等待信号量”等状态。
3. 图形中应能体现事件控制块和事件的对应关系。
4. 可以按事件号、事件类型号、栈尺寸、最近1秒CPU占用率、历史CPU占用率等。

### 对象文件显示

显示object树。

能够按树形结构显示object的名字和主要成员，还要展开显示ObjectPort链表中所有节点。

对于系统定义的object，须显示“represent”指向的实体值。

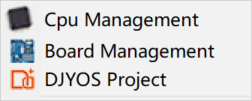
### 信号量显示

显示系统中全部信号量、互斥量的状态，以及空闲信号量控制块的数量。

须显示被他们阻塞的事件，特别是，如果有优先级继承，须显示被阻塞事件的原优先级和现优先级。

# BSP设计向导

在下图中增加“BSP Project”选项和“chip driver”选项



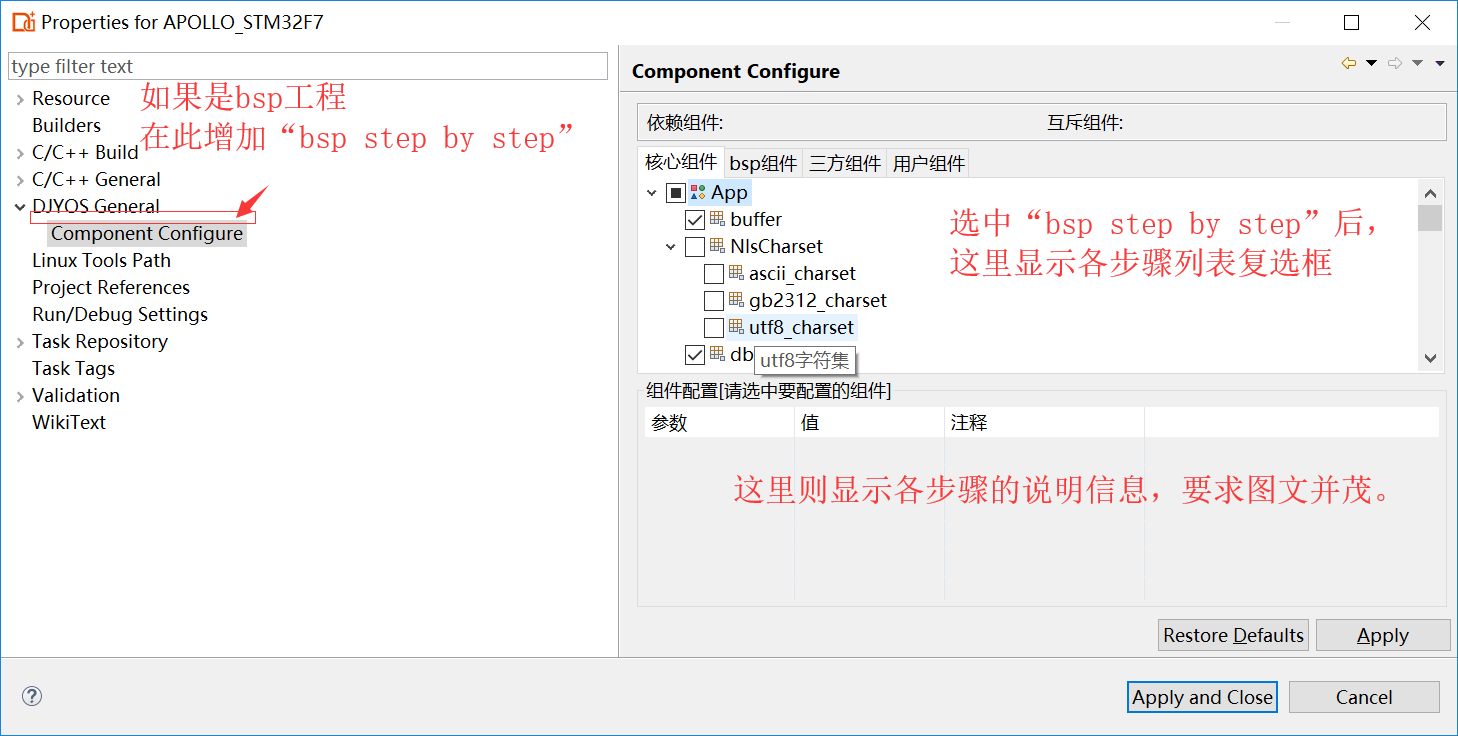
## 创建bsp工程



Bsp工程主要作用是两个：

1. 及时增量式地调试，使新bsp的相关代码，能够step by step地编译和调试。
2. 在工程属性中，提供向导和检查表，引导用户step by step地完成编码。

Bsp 工程需要在工程属性中，界面修改：



## CPU startup代码

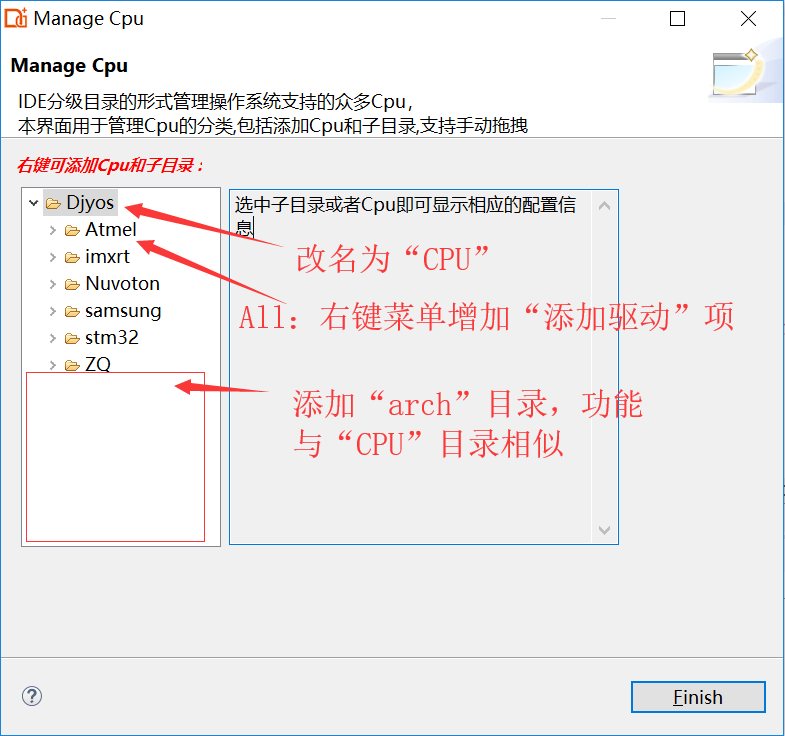
上电boot，是指CPU启动最初的初始化过程，放在boarddrv/user/boardname/startup目录中。

根据创建板件时所选的参考板件与新板件相似度差别，创建板件时自动生成的上电startup代码的完善情况也有区别。

在“bsp step by step”菜单中，startup代码条目如下

* CPU startup
  + CPU启动
  + 主时钟初始化
  + 内存总线初始化
  + cache设置
  + mmu设置

## 改进CPU管理功能



arch添加后，在arch目录中创建相应目录和xml文件，驱动要一个个编写。

“arch”目录的子目录中选中“添加驱动”项时，（若需要）即在该项下创建include和src目录，并提示在这两个目录下添加相应代码。

CPU添加后，外设驱动也要一个个编写。

“CPU”目录的子目录中选中“添加驱动”项时，如果没有src和include目录，则创建之。询问外设名字，创建cpu\_peri\_name.c和cpu\_peri\_name.h两个文件，并打开。文件初始内容来自模板文件。

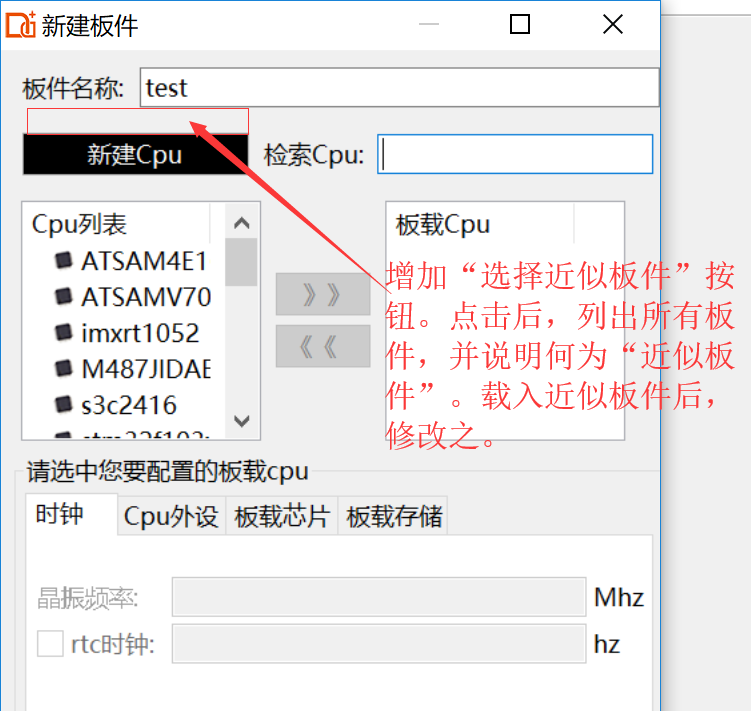
提问用户是否退出界面去编写驱动，选“否”则继续留在此界面，“是”则编辑刚创建的文件。

在“bsp step by step”菜单中，增加arch代码和CPU代码条目如下：

* Arch驱动
  + arch\_feature.h和arch\_stdint.h文件定义
  + 上下文切换相关代码。
  + 异常系统和中断
  + Shell相关的汇编。
  + uS和nS延时代码。
  + tick系统（也可能在CPU驱动里）
* CPU驱动

## 改进板件管理功能

板件管理功能，原来只有配置功能，要增加完成板件驱动代码的功能。



板件近似度排序：CPU相同——>arch相同——>厂商系列近似

如果选择了相似板件，则列出其boarddrv目录下的全部文件，用户勾选中的文件，copy到新板件的目录下。

在“bsp step by step”菜单中，增加boarddrv代码条目如下：

* boarddrv驱动
  + 板件gpio配置。
  + 强制运行Iboot函数
  + 默认网络参数配置。
  + 各外设跟板件相关配置。

## 添加芯片驱动

在这里点选“chip driver”，输入芯片名，检查重名通过后，自动在chipdrv目录下创建chipname目录和chipname.c文件，在chipdrv/include目录下创建chipname.h文件。打开c和h文件，由用户编辑。