# 嵌入式开发软件架构设计

## 架构设计简述

分层思路：功能模块设计与分层

API设计可以分为驱动层和应用层，同时并不是所有程序都调用驱动层API。整个应用中都调用驱动层API会导致应用中驱动调用随处可见，无法移植和最大限度的复用。

**应用层**：把一个应用进行功能模块划分，并对整体结构进行分层，然后设计出功能独立的各个模块（如算法模块、文件库模块、通信库模块等），在模块之上开放公共接口。

**驱动层**：驱动层提供出公共接口供上层调用。各个功能模块可以独立编译（如算法模块纯ANSI C，可在任意平台复用），或者调用驱动层接口（文件库模块调用驱动读写Flash），封装出各个功能独立的可复用的功能模块。

总体划分：硬件驱动层->功能模块层->应用接口层->业务逻辑层->应用层

## 总体规划示意图



图 1 软件架构分层结构示意图

**应用层**：为程序的总体的运行框架，组织调用业务逻辑。可以用某种嵌入式操作系统实现几种任务 。如定时任务，事件处理任务，菜单任务，通信任务。

**业务逻辑层**：如CPU卡处理，个事件处理，，通信记录上传，控制命令下发，参数存储，系统状态监测等。

**应用接口层**：提供公共的API接口供应用接口供上层调用。这些接口也可由下层的功能模块开放出来，应用接口层负责汇总。

**功能模块层**：可以封装不同的功能模块。如算法库，文件库，通信库，显示库，数据库等，向上提供应用接口层的接口，向下调用驱动接口。

**硬件驱动层**：由各个驱动模块组成，向上提供统一的接口。

## 软件架构的特性

### 主要特点

1. 层与层之间不能跨层调用；
2. 模块与模块之间各自独立，无关联，无依赖关系；
3. 模块提供统一的接口供上层调用，模块的内外接口分明；
4. 模块的功能只能增，不能改；
5. 各个功能模块层也可以继续分层，比如接口层、驱动层、硬件层等；

### 遵循一些约定

1. 每个模块提供出的接口要统一，后续只能增，不能改原来的接口；
2. 模块与模块之间相互独立，互不影响，不能相互调用，只能调用它下层接口；
3. 由模块构成层，层与层之间不能跨级调用。如在应用层中不能看到直接调用驱动层的代码；
4. 模块中又可以继续分层，如接口层，驱动层，硬件层；

如果驱动变动了，或者换不同平台，只需更改驱动层，应用层不受影响。

如果功能模块变动了，只需升级功能功能模块，其他的模块不受影响，应用层也不受影响。

按照这种逻辑设计好之后，主要的工作就是在业务逻辑层。应用层则为程序的总体流程和框架，主要调用业务逻辑层实现不同的功能。

## 架构其他内容

实际工作中，很多工程师并未考虑嵌入式软件架构设计，主要的原因如下：

1. 芯片本身的模块化程度已经很高；
2. 是在芯片资源有限。功耗有限等特殊情况下， 进行优化编程；
3. 低端产品，每年都需要降低成本；
4. 软件本身的复杂度不是很高，对代码的复用性要求不是很高；
5. 导致模块化设计、软件架构设计等要求不高；
6. 复用要付出存储空间和时间的代价的，电子产品往往需要和硬件成本相冲突；

## 实际案例

### 项目目标

新产品研发，负责加速度计、NB-IOT、舵机、外置Flash的功能测试；同时输出产品设计概要；

### 初步方案



图 2 架构设计方案一

方案一缺点：序号标记，阅读困难，层次不清楚

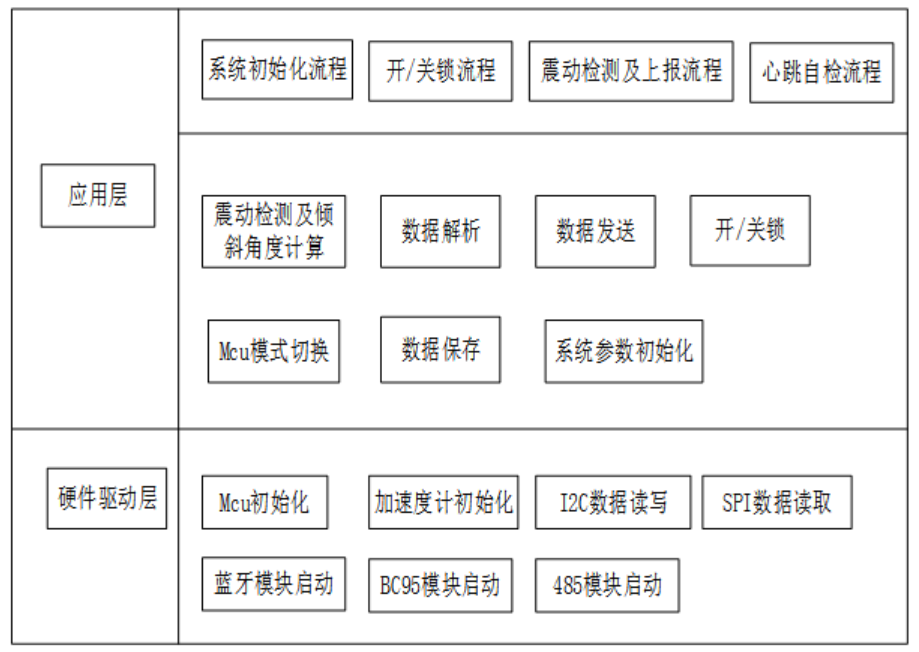


图 3 架构设计方案二

方案二主要存在的问题：

1. 对架构的理解还不是很清晰，既然是做架构设计，那就应该从整体来看，而不是仅仅只是局限于一个模块，或者功能里面。
2. 还是每个层次的理解也还不是很清晰，比如讲MCU的初始化，归于硬件驱动层里面。MCU的初始化，严格意义上来说，是属于流程的一部分了，而不是驱动。比如电脑的开启启动，把这个归于硬件的驱动里面，肯定是属于牛头不对马嘴的。
3. 还有就是各个模块的启动，也是不能属于硬件驱动层的，也都是业务流程的一部分了，都不应该属于驱动层的一部分。
4. 还有就是总线数据的读写，虽然驱动的作用也就是读写，但是数据总线的读写不能写成硬件驱动。
5. 应用层的系统参数初始化，也还是属于流程。
6. 数据的解析和数据的发生，都是属于通信功能里面的，不应该单独独立出来，属于单个的应用

### 设计方案

#### 设计目的

1. 应用的代码逻辑清晰，且避免重复的造轮子；
2. 如果没有好的架构，移植将会是一件很痛苦的事情，因此一个好的架构设计，方便软件的移植；
3. 最大限度的复用；
4. 高耦合低内聚；

#### 设计思路

如何把硬件的驱动和一个功能封装成一个个的模块，然后可以像小朋友搭积木一个，一个个模块可以快速的拼接起来，组成一个个不同的模型。

我们的嵌入式架构思路也是来源于此，即功能模块化设计、分层设计。这个设计和WEB开发的MVC模式类似，都是注重分层设计。

模块化设计：将收集到的需求，进行归类，总结和分析，将这些需求概括为一个个单独的功能，每一个功能，做成一个单独的功能模块。

分层设计一句话不好直接表达，其主要体现在一下几方面：

1. 功能模块对外调用的模块封装成一个个API，将底层驱动做个API以供功能模块调用。（各个功能模块可以独立编译（如通信模块纯ANSI C，可在任意平台复用），或者调用驱动层接口（日志库模块调用了驱动读写Flash），总而言之，言而总之，封装出各个功能独立的可复用的功能模块。
2. API分为驱动层API和应用层API，而不是所有程序都调用驱动层API。(整个应用中都调用驱动层API会导致应用中驱动调用随处可见，无法移植和最大限度的复用)

**总体分 硬件驱动层-->功能模块层-->业务逻辑层-->应用层**

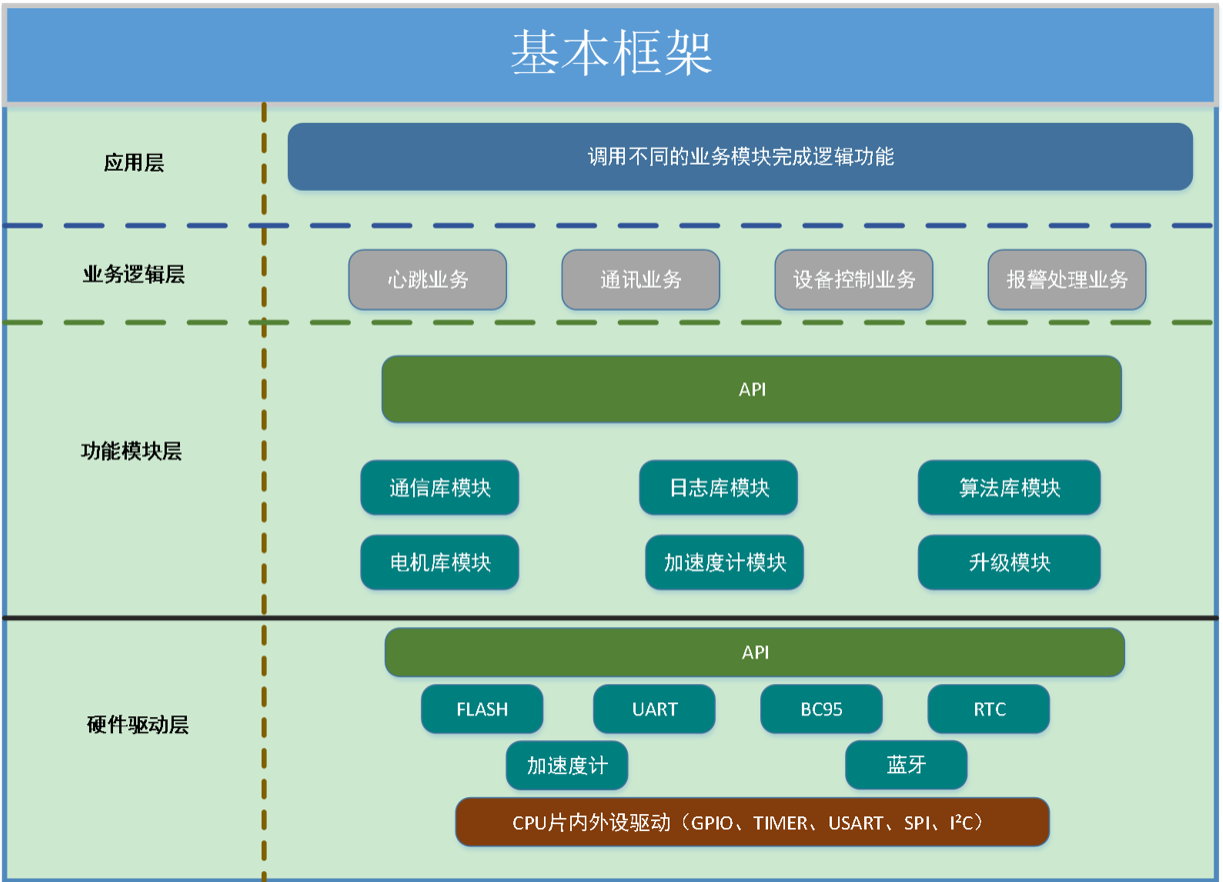


图 4 总体结构示意框图

#### 设计说明

1. **硬件驱动层**

硬件驱动层包含板载硬件资源正常运行所需的所有驱动程序并提供API给功能模块调用。

1. **功能模块层**

功能模块层包括实现具体功能的函数，通过调用驱动层API实现相应功能，同时提供可调用的API给业务逻辑层。

1. **业务逻辑层**

业务逻辑层包括产品整体功能的各个业务流程，通过调用功能模块层的API实现。

1. **应用层**

应用层将各个业务逻辑进行整合调用，完成整个产品的功能。

#### 相关优势

1. 如果驱动变动了，或者换不同平台，只需更改驱动层，应用层不受影响。
2. 如果功能模块变动了，只需升级相应的功能模块，其他的模块不受影响，应用层也不受影响。
3. 按照这种逻辑设计好之后，主要的工作就是在业务逻辑层。应用层则为程序的总体流程和框架，主要调用业务逻辑层实现不同的功能。

## 参考链接

1. <https://www.cnblogs.com/kmust/p/9250263.html>
2. <http://www.cnblogs.com/engraver-lxw/p/7396481.html>