**cComponentStrut设计说明**

仅个人使用

180712V1.2

**目录**

[**cComponentStrut设计说明** I](file:///C:\Users\pxf\Desktop\cComponentStrut\doc\cComponentStrut设计说明.docx#_Toc521850606)

[**目录** i](#_Toc521850607)

[**修订记录：** iii](#_Toc521850608)

[**文档说明：** iii](#_Toc521850609)

[**层级示意：** iv](#_Toc521850610)

[**1.** **一级标题** iv](#_Toc521850611)

[**1.1.** **二级标题** iv](#_Toc521850612)

[**1.1.1.** **三级标题** iv](#_Toc521850613)

[**1.1.1.1.** **四级标题** iv](#_Toc521850614)

[**1.1.1.1.1.** **五级标题** iv](#_Toc521850615)

[**1.1.1.1.1.1.** **六级标题** iv](#_Toc521850616)

[**1.1.1.1.1.1.1.** **七级标题** iv](#_Toc521850617)

[**1.1.1.1.1.1.1.1.** **八级标题** iv](#_Toc521850618)

[**1.1.1.1.1.1.1.1.1.** **九级标题** iv](#_Toc521850619)

[**1.** **引言** - 1 -](#_Toc521850620)

[**1.1.** **编写目的** - 1 -](#_Toc521850621)

[**1.2.** **背景** - 1 -](#_Toc521850622)

[**1.3.** **定义** - 1 -](#_Toc521850623)

[**1.4.** **参考资料** - 1 -](#_Toc521850624)

[**2.** **架构总体设计** - 2 -](#_Toc521850625)

[**2.1.** **要求** - 2 -](#_Toc521850626)

[**2.2.** **整体设计** - 2 -](#_Toc521850627)

[**3.** **详细描述** - 5 -](#_Toc521850628)

[**3.1.** **bsp接口** - 5 -](#_Toc521850629)

[**3.1.1.** **说明** - 5 -](#_Toc521850630)

[**3.1.2.** **要求** - 5 -](#_Toc521850631)

[**3.2.** **abi** - 6 -](#_Toc521850632)

[**3.2.1.** **abi类** - 6 -](#_Toc521850633)

[**3.2.2.** **abi** - 6 -](#_Toc521850634)

[**3.2.3.** **接口抽象** - 6 -](#_Toc521850635)

[**3.2.4.** **组件继承** - 7 -](#_Toc521850636)

[**3.3.** **sch** - 8 -](#_Toc521850637)

[**3.3.1.** **sch整体设计** - 8 -](#_Toc521850638)

[**3.3.2.** **sch类设计** - 9 -](#_Toc521850639)

[**3.3.3.** **sch状态机设计** - 9 -](#_Toc521850640)

[**3.3.4.** **任务组状态机设计** - 10 -](#_Toc521850641)

[**3.4.** **vfb** - 11 -](#_Toc521850642)

[**3.4.1.** **vfb类** - 11 -](#_Toc521850643)

[**3.4.2.** **vfb** - 11 -](#_Toc521850644)

[**3.4.3.** **通信对接** - 11 -](#_Toc521850645)

[**3.5.** **cpn** - 13 -](#_Toc521850646)

[**3.5.1.** **cpn类** - 13 -](#_Toc521850647)

[**3.5.2.** **cpn** - 13 -](#_Toc521850648)

**修订记录：**

|  |  |  |  |
| --- | --- | --- | --- |
| **版本号** | **修订人** | **修订日期** | **修订描述** |
| V1.0 | 潘小飞 | 161009 | 初次生成当前模板 |
| V1.1 | 潘小飞 | 170129 | 每级标题增加定格数字 |
| V1.2 | 潘小飞 | 180712 | 标题不进行缩进并且每级标题都只使用数字 |
|  |  |  |  |

**文档说明：**

1) 文档用于记录自己写的一些文字。

2) 文档封皮不做要求，不同内容可以有不同形式。

3) 正文内容要求使用仿宋小四。

4） 小节标题使用仿宋一号标题一定要加粗。

5) 二级标题以及其他标题使用仿宋**小一**他使用小四。

6） 其他标题依次类推，标题一定要**加粗**。

7） 段落后增加0.35行行距。

**层级示意：**

**以下为层级标题示意，不能删除。删除之后在后面的编排显示不了层级标题。**

1. **一级标题**
   1. **二级标题**
      1. **三级标题**
         1. **四级标题**
            1. **五级标题**

**六级标题**

**七级标题**

**八级标题**

**九级标题**

1. **引言**
   1. **编写目的**

在嵌入式开发中，每个MCU开发都需要重新写自己的代码，特别是不同平台的MCU之间差别还是非常的大。

目前想设计一种架构可以把这种开发的差别进行消除，同时也希望所有的功能进行组件化，使代码开发难度大大简化，一个组件只提供一个功能，需要的组件才拿进来进行编译。

* 1. **背景**

当前C语言可以实现面向对象了，这对于开发架构来说可以省去很多麻烦。所有的代码实现思考角度也少了很多，实际操作简单了很多。

* 1. **定义**

无

* 1. **参考资料**

1. **架构总体设计**
   1. **要求**

对于架构来说，不管是什么样的硬件，都需要可以实现相应的代码。对于设计原则来说主要是以下几个方面。

1. 架构需要与硬件实现完全隔离。
2. 所有功能都由于组件来进行实现，并且一个组件只有一个功能。实际项目根据需要选择功能组件。
3. 所有组件之间的通信都是通过虚拟总线来进行实现。
4. 组件功能的发挥依赖于任务调度定时进行调度。
5. 所有组件都需要做诊断功能，任务调度需要定时去查询诊断状态（可以做一个服务组件），并做出相应的处理。

实现了这几个宏观要求，在架构中运行的程序可以做到快速开发，以及功能的升级优化。

* 1. **整体设计**

架构的设计也是按照设计要求进行实现，整体设计如下。



图2.2.1 架构总体设计

程序开发，主体进行架构分层，分为以下几个层次：

* uC硬件层

纯硬件，外在环境决定；

* bsp板级支持包基础硬件控制层

最基础硬件控制，与硬件存在很大的关联；但有一个则重点，必须考虑abi接口层中相关函数，需要形式一致；

* abi抽象接口层

按功能进行分类进行抽象，每个类型确定之后，不会再变更；如可以对GIO相关接口进行抽象，可以对ADC接口进行抽象；

由于硬件实际对应关系，抽象类是可以有多个实现的；

* sch组件任务调度器

实现所有组件按节拍进行调度，使组件功能发挥作用；

* cpn功能组件
* 服务组件

硬件功能并且有多组件进行控制，多数情况下会把相应的功能做成服务组件；

功能控制比较复杂，为了简化开发难度，可以把相关功能做成服务组件，用户通信功能组件通过vfb进行服务控制；

* 协议组件

协议组件一般来说会与两个以上组件进行通信，基中一个大部分情况是服务组件；

协议组件只是一个中间过程，根据输入给出输出；

* 用户通信功能组件

根据输入进功能控制，给出功能控制到vfb中；

通信功能组件是趋向，因为在这个组件中无关的或者复杂的功能可以交给别的组件来做，需要实现的只是相关的控制逻辑；

* 用户硬件功能组件

因为控制功能简单，所有的控制功能用户一步到位，不用其他组件对功能进行拆分；

* 用户功能组件

与服务组件差不多，此类组件应尽可能地少，其功能可以由服务组件进行替代；目前这个算是作为用户功能控制的一个组件预留；

* vfb组件通信虚拟总线

vfb中进行传递的是消息，消息类型只有两种：变量型和函数通知型。

变量型传递很简单，只要给到vfb就可以不管了；

函数型消息一般要通过队列进行传递，消息接收方要有接收队列，消息产生后会立即给到接收方；

变量型消息是异步传递，只有当接收方到调度节拍到了，才会对消息进行接收，并且在此传递的过程中，因通信组件调度节拍不一样，有存丢失一部分的可能；

函数型消息是同步传递，只要产生了，立即就到达了接收方队列，实时性很好，同时也不会有丢失的可能；

1. **详细描述**
   1. **bsp接口**

bsp的全称是Board Support Package。bsp是一些基础的函数包，为操作硬件提供基本的功能。

* + 1. **说明**

bsp接口由实际硬件决定，接口的多少与以及有无得看硬件是否支持。

* + 1. **要求**

bsp接口必须以abi类中函数形式为基准，如果达不到要求，则必须在现有接口上再套一层abi类中函数形式。

bsp只有接口，不存在类，因为本层是最基础的底层，关心的只有具体操作，抽象化由abi接口进行操作。

* 1. **abi**

abi指的是abstract interface。一般说abi指的是abi层实际抽象出来的接口。

abi层包含的两种类型，abi接口和abi类。abi接口是对abi类实现，是abi类的实例。很多情况下，一个abi类不是只有一个abi接口，abi接口由硬件提供的支持决定。

* + 1. **abi类**

abi类是对一类硬件功能的抽象。如GIO可以抽象为一个类，ADC各通道可以抽象为一个类，CAN模块功能可以抽象为一个类。

抽象，主要是为了，一，统一类型；二，相关功能结合到一块。对抽象类进行设计，遵循这两个原则，设计出来的接口，基本就可以做到一致性了。

* + 1. **abi**

abi接口是对abi类的实现。除了那些被组合到一个abi接口中的bsp接口，其余有多少bsp接口就要有多少abi接口。abi接口是bsp接口的抽象，其个数与bsp接口应是一致的。

* + 1. **接口抽象**

一般情况下，一个abi接口只会对应一个实际bsp接口；abi接口与bsp接口实际可以\*<-->\*地进行抽象；为了尽最大的可能减少控制出错，推荐的方式是abi接口与bsp接口以1<-->\*进行抽象，可以实现bsp接口的原子操作。



图3.2.3.1接口抽象

如果abi接口有多个component进行操作，最好是建立一个服务组件component，把abi接口继承到服务组件中。

* + 1. **组件继承**

abi接口与component的关系是继承，component类会继承abi类。一般情况下，一个cpn只会对应一个abi接口；cpn与abi接口实现可以\*<-->\*地进行对接。



图3.2.4.1组件继承

如果某个abi接口会被多个component进行对接，最好是建立一个服务组件component，把abi接口继承到服务组件中；其他组件通过vfb来进行通信访问。

* 1. **sch**

sch全称是schedular。sch负责的功能就是使各个组件按设置的节拍进行调度，实现组件的功能控制。

* + 1. **sch整体设计**

任务调度器，最主要的功能是通过任务调度状态机去管理任务组。



图3.3.1.1任务调度状态机

目前由于C语言的面向对象功能很强大，调度器建立的方式是面向对象。通过对象来管理调度状态机。

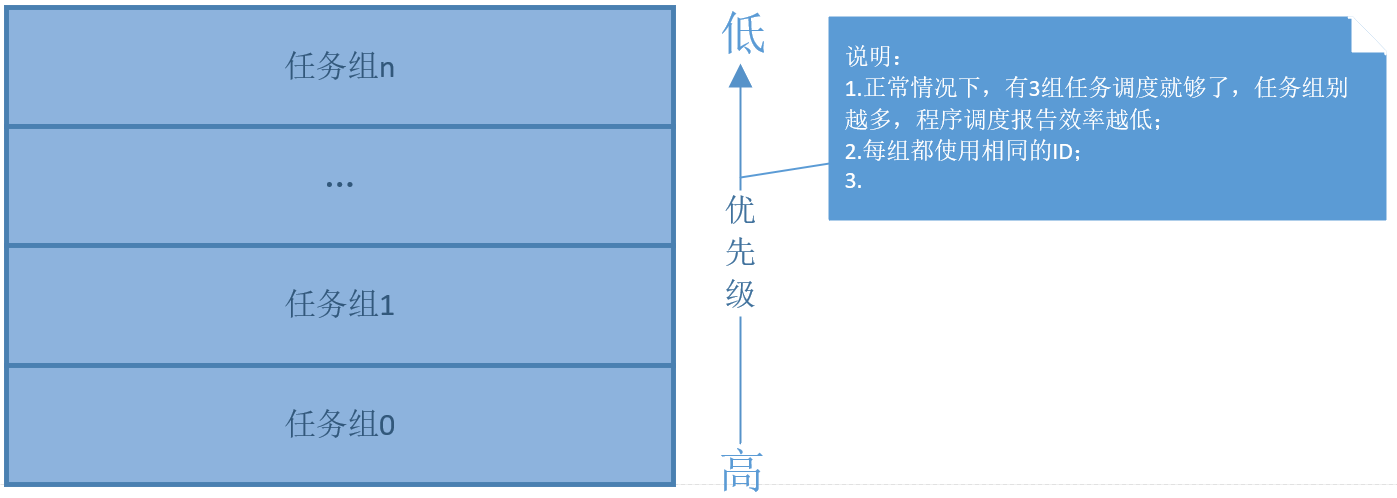


图3.3.1.2任务调度任务组



图3.3.1.3任务组设计

每个任务组都很符合状态机的形式，所以单个任务组也是通过状态机来进行管理的。



图3.3.1.4整体实现设计结构

实际实现过程由调度器类控制两级状态机。

* + 1. **sch类设计**
    2. **sch状态机设计**
    3. **任务组状态机设计**

* 1. **vfb**

vfb全称是visual function bus。每个vfb实现的功能是某两个cpn之间的所有通信。

* + 1. **vfb类**

vfb类依据组件进行划分为3类，组件输入类vfbI，组件输出类vfbO，组件管理类vfbM。vfbI和vfbO是组件的输入输出进行抽象，在组件生成时，必须把对应功能配置好。vfbM由具体项目使用时进行搭配，设置相应的连接。

* + 1. **vfb**

vfb指的是各类的实例化。vfb类的3种类型每个类只对进行一次实例化。由于通信连接的关系，继承关系可以会复杂一些。

* + 1. **通信对接**

vbfM是以vfbO作为关注对象的。关心对所有的输出进行分配。



图3.4.3.1通信对接

vfb与cpn的对接关系是2<-->1，一个cpn会自动抽象出2个vfb接口，一个作为输入，一个作为输出。vfb中以cpn输出抽象接口建立vfb管理类，vfbM与cpn的对接关系是1<-->1。

在vfbM中实现通信的连接，通信连接与实际项目中使用情况有关系。vfbM需要根据项目实际情况建立连接。

vfbM与vfbO的对应关系是1<-->1。

vfbM与vfbI的对应关系是1<-->1\*。

* 1. **cpn**

cpn的全称是component。组件是程序的功能部分，不管是系统级策略还是服务功能，都是可以制作成组件，而实际功能的独立开发。

* + 1. **cpn类**

cpn按功能划分有5类，服务组件、协议组件、用户通信功能组件、用户硬件功能组件、用户功能组件。本质上其实只有3种类型服务组件（用户功能组件）、用户通信功能组件（协议组件）、用户硬件功能组件。但为了使不同的组件明确对应不同的功能，还是需要定义组件类型为5类。



图3.5.1.1组件分类

在这里有一个很重要的问题是在设计cpn类，需要把vfbI及vfbO虚拟通信抽象类给设计出来，并与组件放置在一起。

* + 1. **cpn**

cpn是对cpn类的实例化。每个cpn类只能有一个实例。

//TODO

abi/sch/vfb/cpn其余待实际编码设计完善。