**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#_Toc523585507)

[**目录** i](#_Toc523585508)

[**修订记录：** iii](#_Toc523585509)

[**文档说明：** iii](#_Toc523585510)

[**层级示意：** iv](#_Toc523585511)

[**1.** **一级标题** iv](#_Toc523585512)

[**1.1.** **二级标题** iv](#_Toc523585513)

[**1.1.1.** **三级标题** iv](#_Toc523585514)

[**1.1.1.1.** **四级标题** iv](#_Toc523585515)

[**1.1.1.1.1.** **五级标题** iv](#_Toc523585516)

[**1.1.1.1.1.1.** **六级标题** iv](#_Toc523585517)

[**1.1.1.1.1.1.1.** **七级标题** iv](#_Toc523585518)

[**1.1.1.1.1.1.1.1.** **八级标题** iv](#_Toc523585519)

[**1.1.1.1.1.1.1.1.1.** **九级标题** iv](#_Toc523585520)

[**1.** **引言** - 1 -](#_Toc523585521)

[**1.1.** **编写目的** - 1 -](#_Toc523585522)

[**1.2.** **背景** - 1 -](#_Toc523585523)

[**1.3.** **定义** - 1 -](#_Toc523585524)

[**1.4.** **参考资料** - 1 -](#_Toc523585525)

[**2.** **架构总体设计** - 2 -](#_Toc523585526)

[**2.1.** **要求** - 2 -](#_Toc523585527)

[**2.2.** **整体设计** - 2 -](#_Toc523585528)

[**2.3.** **组件部署** - 5 -](#_Toc523585529)

[**2.3.1.** **初始化组件** - 5 -](#_Toc523585530)

[**2.3.2.** **组件调度服务组件** - 5 -](#_Toc523585531)

[**2.3.3.** **异步服务组件** - 6 -](#_Toc523585532)

[**2.3.4.** **服务组件** - 7 -](#_Toc523585533)

[**2.3.5.** **用户功能组件** - 8 -](#_Toc523585534)

[**2.3.6.** **协议控制组件** - 9 -](#_Toc523585535)

[**2.3.7.** **通信控制组件** - 10 -](#_Toc523585536)

[**2.4.** **时序设计** - 11 -](#_Toc523585537)

[**2.4.1.** **初始化时序** - 11 -](#_Toc523585538)

[**2.4.2.** **组件调度时序** - 11 -](#_Toc523585539)

[**2.4.3.** **异步服务时序** - 12 -](#_Toc523585540)

[**2.4.4.** **服务时序** - 13 -](#_Toc523585541)

[**2.4.5.** **用户功能时序** - 14 -](#_Toc523585542)

[**2.4.6.** **协议控制时序** - 15 -](#_Toc523585543)

[**2.4.7.** **通信控制时序** - 16 -](#_Toc523585544)

[**3.** **详细描述** - 18 -](#_Toc523585545)

[**3.1.** **bsp接口** - 18 -](#_Toc523585546)

[**3.1.1.** **说明** - 18 -](#_Toc523585547)

[**3.1.2.** **要求** - 18 -](#_Toc523585548)

[**3.2.** **abi** - 19 -](#_Toc523585549)

[**3.2.1.** **abi类** - 19 -](#_Toc523585550)

[**3.2.2.** **abi** - 19 -](#_Toc523585551)

[**3.2.3.** **接口抽象** - 19 -](#_Toc523585552)

[**3.2.4.** **组件继承** - 20 -](#_Toc523585553)

[**3.3.** **cpnInit** - 21 -](#_Toc523585554)

[**3.3.1.** **cpnInit组件** - 21 -](#_Toc523585555)

[**3.4.** **casSch** - 23 -](#_Toc523585556)

[**3.4.1.** **casSch整体设计** - 23 -](#_Toc523585557)

[**3.4.2.** **sch类设计** - 24 -](#_Toc523585558)

[**3.4.3.** **sch状态机** - 25 -](#_Toc523585559)

[**3.4.3.1.** **状态机设计** - 25 -](#_Toc523585560)

[**3.4.3.2.** **状态机代码设计** - 25 -](#_Toc523585561)

[**3.4.4.** **任务组** - 26 -](#_Toc523585562)

[**3.4.4.1.** **任务组设计** - 26 -](#_Toc523585563)

[**3.4.4.2.** **任务组代码设计** - 27 -](#_Toc523585564)

[**3.5.** **casX** - 29 -](#_Toc523585565)

[**3.5.1.** **casX组件** - 29 -](#_Toc523585566)

[**3.5.2.** **casX组件实例化** - 29 -](#_Toc523585567)

[**3.6.** **csX** - 31 -](#_Toc523585568)

[**3.6.1.** **csX组件** - 31 -](#_Toc523585569)

[**3.6.2.** **csX组件实例化** - 31 -](#_Toc523585570)

[**3.7.** **cufX** - 33 -](#_Toc523585571)

[**3.7.1.** **cufX组件** - 33 -](#_Toc523585572)

[**3.7.2.** **cufX组件实例化** - 33 -](#_Toc523585573)

[**3.8.** **cpcX** - 35 -](#_Toc523585574)

[**3.8.1.** **cpcX组件** - 35 -](#_Toc523585575)

[**3.8.2.** **cpcX组件实例化** - 35 -](#_Toc523585576)

[**3.9.** **cccX** - 36 -](#_Toc523585577)

[**3.9.1.** **cccX组件** - 36 -](#_Toc523585578)

[**3.9.2.** **cccX组件实例化** - 36 -](#_Toc523585579)

[**3.10.** **vfb** - 37 -](#_Toc523585580)

[**3.10.1.** **vfb类** - 37 -](#_Toc523585581)

[**3.10.2.** **vfb实例化** - 37 -](#_Toc523585582)

[**3.10.3.** **通信对接** - 37 -](#_Toc523585583)

[**4.** **其他事项** - 39 -](#_Toc523585584)

[**4.1.** **脚本开发** - 39 -](#_Toc523585585)

**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组件定时进行调度。
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 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(board support package)

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(abstract interface)

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，接口类是可以有多个实现的；

* cpnInit初始化组件

实现所有组件组件及接口类初始化；这是一个特殊的组件，每个芯片程序都必须要有，需要用其完成初始化；

* casSch任务调度服务组件

实现所有组件按节拍进行调度，使组件功能发挥作用；本组件是异步服务组件的一个特殊组件；因程序需要运行，必须要有这样的组件组织程序运行结构；

* cpn功能组件(component)
* casX**\***异步服务组件(component asynchronous service)

硬件提供触发功能，多数情况下，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异步服务组件；

异步服务组件会提供很多功能，可以通过虚拟总线把相应的功能给到对应的组件，进行服务控制；

异步服务组件通常是向其他组件进行功能输出的组件，一般来说其输入是硬件；当然也有可能其输出对硬件进行回馈的情况。

* csX**\***服务组件(component service)

服务组件是对硬件进行控制，不管是读取某些状态还是控制某个输出，都可以理解为对硬件的输出。因为只有在某个点才会进行操作，并且操作是由组件进行控制的。

服务组件输出一般来自协议组件或者通信控制组件，输出是指向硬件的。

* cufX**\***用户功能组件(component user function)

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，算是给硬件控制开了一个后门，某些功能不进行组件进行拆解，粗暴地直接进行控制；

* cpcX**\***协议控制组件(component protocol control)

协议组件一般来说会与两个以上组件进行通信;典型的两种情况是，csX🡪cpcX🡪cccX；cccX🡪cpcX🡪csX；

协议组件只是一个中间过程，根据输入给出输出；协议组件是与硬件完全实现隔离的，按协议由输入给出输出；

目前架构设计主要是以单芯片为主的，但是也可以多芯片进行整体设计，此时就需要编码和解码的协议组件实现不同芯片间的通信控制组件和服务组件间的联系。

* cccX**\***通信控制组件(component communication control)

根据输入进功能控制，给出功能控制输出；其形式与协议组件比较类似，与硬件完全实现隔离；

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；

* vfb组件通信虚拟总线(virtual function bus)

vfb中进行传递的是消息，消息可以是任意内容，只要需要都可以进行传递。比如组件的初始化，某个任务的添加，某个变量等等；

虚拟总线分为3部分，vfbI/vfbO/vfbM；其中vfbI/vfbO与组件联系在一起，vfbM独立于组件实现组件输出的管理。

* vfbI虚拟总线输入(virtual function bus input)

虚拟总线输入是当前组件从总线上得到的输入，没有指定的从哪个组件给出来的概念，只要某个组件的虚拟总线管理有获得虚拟总线输入的实例接口，就可以实现对本组件的输入。

* vfbO虚拟总线输出(virtual function bus output)

虚拟总线输出是当前组件给到总线消息，没有指定给到哪个组件，输出只把消息输出到总线上。

* vfbM虚拟总线管理(virtual function bus manage)

虚拟总线管理是对当前组件的虚拟总线输出给到总线消息进行管理，输出只把消息输出到总线上，管理实际输出的再分配。

* 1. **组件部署**
     1. **初始化组件**



图 2 初始化组件

* + 1. **组件调度服务组件**



图 3 组件调度服务组件

* + 1. **异步服务组件**



图 4 异步服务组件

* + 1. **服务组件**



图 5 服务组件

* + 1. **用户功能组件**



图 6 用户功能组件

* + 1. **协议控制组件**



图 7 协议控制组件

* + 1. **通信控制组件**



图 8 通信控制组件

* 1. **时序设计**
     1. **初始化时序**



图 9 初始化时序

* + 1. **组件调度时序**



图 10 组件调度时序

* + 1. **异步服务时序**



图 11 异步服务时序

* + 1. **服务时序**



图 12 服务时序

* + 1. **用户功能时序**



图 13 用户功能时序

* + 1. **协议控制时序**



图 14 协议控制时序

* + 1. **通信控制时序**



图 15 通信控制时序

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 16 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 17 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vfbI来进行通信访问。

* 1. **cpnInit**

cpnInit指的是component initial。cpnInit可以认为是一个特殊的组件。组件功能提供两个方向的传递。一是abiInit，另外一个是vfbIcasSchInit（包含在vfbIcasSch中）。



图 18 cpnInit消息传递

* + 1. **cpnInit组件**

cpnInit组件是为空的，它只包含一个初始化函数。

然后就是输出类，也是为空的，只有一个输出初始化；再到管理类，空的，只有一个初始化函数；最后，初始化传递到抽象类及casSch的输入类的初始化。

抽象类初始化只包含casX/csX/cufX三类组件对应的抽象接口初始化。其中csX类型组件是最常见的；casX一般只作为特殊功能的，如需要线程类功能才会出现此组件；cufX类类型组件，只作为用户预留功能，可作为补丁性质的功能控制，一般会把其拆解为csX及cccX进行控制。

casSch的输入类的初始化对组件自身进行初始化后，其输出的管理类的初始化会管理在本MCU上运行的所有组件的初始化，由其传递到各组件的输入初始化，然后完成对本组件的初始化。



图 19 程序运行层次

* 1. **casSch**

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。



图 20 casSch消息传递

* + 1. **casSch整体设计**

作为整个芯片程序各组件管理调度来说，设计还是很简单的，主体过程如下图。



图 21 casSch整体设计

组件建立sch类，然后通过sch类管理sch状态机，状态机控制任务组实现各任务的调度。

* + 1. **sch类设计**

**CL**(casSch){

hcasSch self;

hcasSch (\*init)(hcasSch cthis, hstaAct smcasSch, hvfbOcasSch vfbOcasSch, hmeasure time);

// 组件运行状态机

smcasSchRec smcasSchRec;

hstaAct smcasSch;

// 组件运行相关参数

uint32 usage; // cpu使用率，如112表示11.2%

uint32 taskTime; // 当前任务执行时间，与时间测量结果一致

// 组件运行功能函数

**void** (\*run)(hcasSch t);

**void** (\*tick)(hcasSch t);

int16 (\*delay)(hcasSch t, uint32 \*tick); // 节拍延时，最大支持32767个节拍，0延时已到，非0延时未到

// 组件输入功能函数

**void** (\*addTask)(hcasSch t, int16 id, **void**(\*schTask)(**void**), uint16 prdTick, uint16 startTick);

**void** (\*delTask)(hcasSch t, int16 id, **void**(\*schTask)(**void**));

// 组件输出功能函数

**void** (\*tickOut)(hcasSch t);

**void** (\*err)(hcasSch t, herrCode code);

// 注入组件输出类及其他功能类

**INJ**(vfbOcasSch, vfbOcasSch);

**INJ**(measure, time);

};

包含任务的功能函数以及管理组件状态机。

* + 1. **sch状态机**
       1. **状态机设计**



图 22 状态机状态跳转设计

* + - 1. **状态机代码设计**

状态声明

/\*组件状态机状态

\* 有两个状态init/default，状态list不显示，默认给出

\* 状态list需根据实际应用进行重写，list修改后相应状态函数得对应修改

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**#define** **SM\_CASSCH\_STA\_LIST**(\_) \

\_(smcasSch, update)\

\_(smcasSch, execute)

状态机声明

/\*组件状态机声明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**SMDC**(*smcasSch*, **SM\_CASSCH\_STA\_LIST**){

sta next;

taskGroups taskGroups;

// 注入组件类

**void** \*casSch;

};

状态机定义

/\*组件状态机定义

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**SMDF**(*smcasSch*, **SM\_CASSCH\_STA\_LIST**);

* + 1. **任务组**
       1. **任务组设计**

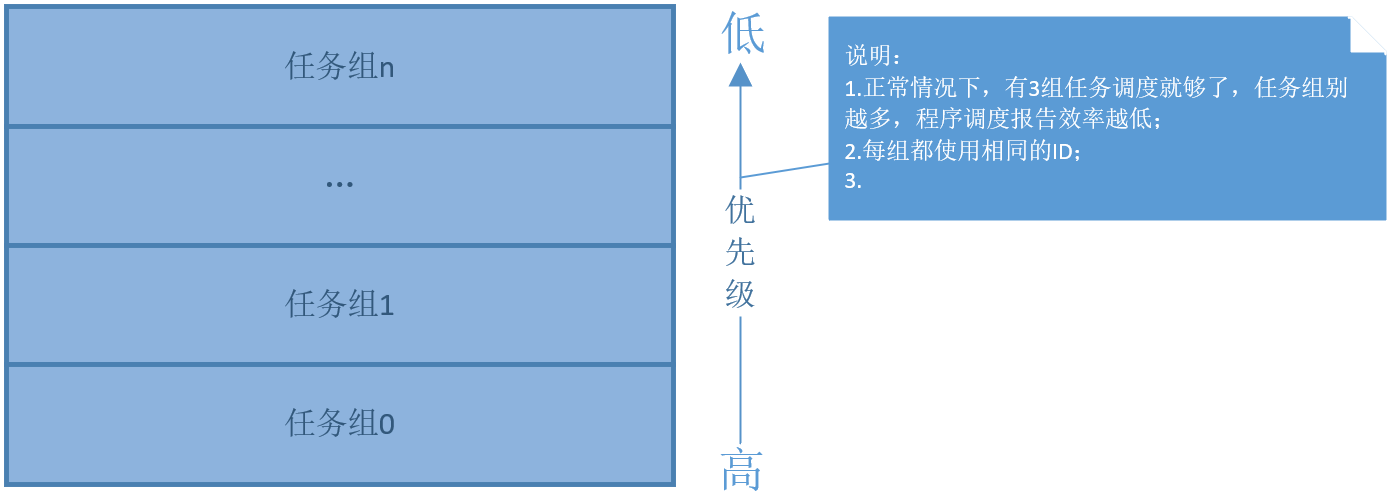


图 23 任务组统一管理



图 24 任务组内部设计



图 25 任务形式

* + - 1. **任务组代码设计**

任务组声明

/\*任务管理组声明

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**typedef** **struct**{

int16 taskGroupNum;

uint16 tick;

uint16 startTick[**SCH\_TASK\_GROUP\_NUM**][32]; // 调度周期起始节拍

uint16 prdTick[**SCH\_TASK\_GROUP\_NUM**][32]; // 周期调度节拍

uint32 actMask[**SCH\_TASK\_GROUP\_NUM**]; // 激活请求任务掩码

uint32 taskMask[**SCH\_TASK\_GROUP\_NUM**]; // 已分配任务掩码

schTask taskGroup[**SCH\_TASK\_GROUP\_NUM**][32]; // 任务组

} taskGroups;

* 1. **casX**

cas全称是component asynchronous service。异步服务组件主要是因为硬件有中断之类的线程，从而提供一个异步服务组件。



图 26 异步服务组件消息传递

* + 1. **casX组件**

组件依据功能进行划分为4类，组件输入类vfbIcasX，组件类casX，组件输出类vfbOcasX，组件管理类vfbMcasX。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. **casX组件实例化**

casXA是对casX类的实例化。每个组件类只能有一个实例。组件中的4个类都会有各自对应的实例vfbIcasXA/casXA/vfbOcasXA/vfbMcasXA。

* 1. **csX**

cas全称是component service。服务组件主要是对硬件功能进行输出操作。一般来说，对硬件进行基本控制的组合会设置为一个服务组件。服务组件只根据输入的指令进行操作，组件内部并不存在策略逻辑。



图 27 服务组件消息传递

* + 1. **csX组件**

组件依据功能进行划分为4类，组件输入类vfbIcsX，组件类csX，组件输出类vfbOcsX，组件管理类vfbMcsX。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. **csX组件实例化**

csXA是对csX类的实例化。每个组件类只能有一个实例。组件中的4个类都会有各自对应的实例vfbIcsXA/csXA/vfbOcsXA/vfbMcsXA。

* 1. **cufX**

cas全称是component user function。用户功能组件主要是用户对硬件功能进行自定义输出操作。其功能与服务组件类似，一般来说，对硬件进行基本控制的组合会设置为一个服务组件。用户功能组件在程序开发过程中应尽量少使用，此功能相当使用后门功能，不利于程序进行组件化规范。



图 28 用户功能组件消息传递

* + 1. **cufX组件**

组件依据功能进行划分为4类，组件输入类vfbIcufX，组件类cufX，组件输出类vfbOcufX，组件管理类vfbMcufX。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. **cufX组件实例化**

cufXA是对cufX类的实例化。每个组件类只能有一个实例。组件中的4个类都会有各自对应的实例vfbIcufXA/cufXA/vfbOcufXA/vfbMcufXA。

* 1. **cpcX**

cpc全称是component protocol control。协议控制组件主要是对制定的功能协议进行组件化，独立于所有功能策略和功能控制。协议控制组件是独立于硬件的，只要给入输入，就会按协议给出输出。



图 29 协议控制组件消息传递

* + 1. **cpcX组件**

组件依据功能进行划分为4类，组件输入类vfbIcpcX，组件类cpcX，组件输出类vfbOcpcX，组件管理类vfbMcpcX。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. **cpcX组件实例化**

cpcXA是对cpcX类的实例化。每个组件类只能有一个实例。组件中的4个类都会有各自对应的实例vfbIcpcXA/cpcXA/vfbOcpcXA/vfbMcpcXA。

* 1. **cccX**

cpc全称是component communication control。通信控制组件主要是对某类功能策略控制进行组件化。通信控制组件是独立于硬件的，只要有输入，就会按已制定的策略给出输出。



图 30 通信控制组件消息传递

* + 1. **cccX组件**

组件依据功能进行划分为4类，组件输入类vfbIcccX，组件类cccX，组件输出类vfbOcccX，组件管理类vfbMcccX。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. **cccX组件实例化**

cccXA是对cccX类的实例化。每个组件类只能有一个实例。组件中的4个类都会有各自对应的实例vfbIcccXA/cccXA/vfbOcccXA/vfbMcccXA。

* 1. **vfb**

vfb全称是visual function bus。所有组件间的消息传递都是通过vfb来进行的。vfb中传递的消息是没有类型限制的，只要需要都可以通过其进行。

* + 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实例化**

vfb实例化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，vfbM必须继承vfbO。

* + 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 31 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. **其他事项**
   1. **脚本开发**

进行程序开发，最无聊的是对一些东西进和拷贝然后进行修改，一是这样的工作就是纯体力劳动，二是这样的操作很有可能出现某些修改遗漏的地方，特别是修改内容多起来之后。

目前想要自己开发一套开发工具很不现实，一没精力，二时间成本太大，目前选择使用python作为脚本来开发需要的功能，可以大大缩减开发难度，那个人的操作就会强很多了。

目前所有类型的组件模板生成已支持，使用以下指令可以生成各种类型组件：

|  |  |  |
| --- | --- | --- |
| 支持组件 | cas、cs、cuf、cpc、ccc | 生成组件名称 |
| 指令写法 | python cpnshell.py cpntype cpnname | cpntypecpnname |
| cas生成示例 | python cpnshell.py cas PwmServ | casPwmServ |
| cs生成示例 | python cpnshell.py cs LedServ | csLedServ |
| cuf生成示例 | python cpnshell.py cuf LedFunc | cufLedFunc |
| cpc生成示例 | python cpnshell.py cpc LedCpc | cpcLedCpc |
| ccc生成示例 | python cpnshell.py ccc LedCtrl | cccLedCtrl |
|  |  |  |

脚本代码：

