**CpnFramework运行架构**

仅个人使用

|  |  |  |
| --- | --- | --- |
| 文件状态 | 文件标识 |  |
| [√] 草稿  [ ] 正在修改  [ ] 正式发布 | 当前版本： | V1.0 |
| 作 者： | 潘小飞 |
| 审 核： | 潘小飞 |
| 起止日期： | 191222-- |

**目录**

[**CpnFramework运行架构** I](file:///D:\MyFiles\github\CpnFramework\Doc\CpnFramework运行架构_200809.docx#_Toc47874581)

[**目录** ii](#_Toc47874582)

[**修订记录：** iv](#_Toc47874583)

[**文档说明：** iv](#_Toc47874584)

[1. 引言 - 1 -](#_Toc47874585)

[1.1. 编写目的 - 1 -](#_Toc47874586)

[1.2. 背景 - 1 -](#_Toc47874587)

[1.3. 定义 - 1 -](#_Toc47874588)

[1.4. 参考资料 - 1 -](#_Toc47874589)

[2. 整体设计 - 2 -](#_Toc47874590)

[2.1. 整体架构 - 2 -](#_Toc47874591)

[2.2. RTE设计 - 2 -](#_Toc47874592)

[2.2.1. set/get接口设计 - 3 -](#_Toc47874593)

[2.2.2. set/get接口配置 - 4 -](#_Toc47874594)

[2.2.3. serv/trig接口设计 - 6 -](#_Toc47874595)

[2.2.4. serv/trig接口配置 - 6 -](#_Toc47874596)

[2.3. CPN组件设计 - 9 -](#_Toc47874597)

[2.3.1. 组件信号配置 - 9 -](#_Toc47874598)

[2.3.1.1. RTE信号 - 9 -](#_Toc47874599)

[2.3.1.1.1. RTE信号配置 - 10 -](#_Toc47874600)

[2.3.1.1.1.1. RteSigs.h - 10 -](#_Toc47874601)

[2.3.1.1.1.2. RteAsynSigs.h - 10 -](#_Toc47874602)

[2.3.1.1.1.3. RteAsynSigs.c - 12 -](#_Toc47874603)

[2.3.1.1.1.4. RteSynSigs.h - 13 -](#_Toc47874604)

[2.3.1.1.1.5. RteSynSigs.c - 15 -](#_Toc47874605)

[2.3.1.2. 组件信号 - 16 -](#_Toc47874606)

[2.3.1.2.1. CpnSampleSig.h - 16 -](#_Toc47874607)

[2.3.1.2.2. CpnSampleSig.c - 19 -](#_Toc47874608)

[2.3.1.2.3. CpnSampleCfg.h - 20 -](#_Toc47874609)

[2.3.1.3. 脚本生成 - 21 -](#_Toc47874610)

[2.3.2. CpnSch调度组件 - 21 -](#_Toc47874611)

[2.3.2.1. 任务组 - 21 -](#_Toc47874612)

[2.3.2.2. 任务等级 - 22 -](#_Toc47874613)

[2.3.2.3. 任务调度结构 - 23 -](#_Toc47874614)

[2.3.2.4. 组件信号 - 24 -](#_Toc47874615)

[2.3.2.5. 组件参数配置 - 25 -](#_Toc47874616)

[2.3.3. CpnErr错误处理组件 - 26 -](#_Toc47874617)

[2.4. ABI设计 - 27 -](#_Toc47874618)

[2.5. BSP结构 - 27 -](#_Toc47874619)

[3. 组件设计 - 28 -](#_Toc47874620)

[3.1. 脚本说明 - 28 -](#_Toc47874621)

[3.1.1. C模板 - 28 -](#_Toc47874622)

[3.1.2. OOPC对象 - 28 -](#_Toc47874623)

[3.1.3. SM状态机 - 29 -](#_Toc47874624)

[3.1.4. CPN新建 - 29 -](#_Toc47874625)

[3.1.5. CPN操作 - 30 -](#_Toc47874626)

[3.2. 组件组成 - 31 -](#_Toc47874627)

[3.2.1. 组件信号 - 31 -](#_Toc47874628)

[3.2.2. 组件对象 - 31 -](#_Toc47874629)

[3.3. 组件匹配 - 31 -](#_Toc47874630)

**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 |
| V1.3 | 潘小飞 | 180910 | 删除标题层级示意结构，更新标题自动编号 |
| V1.4 | 潘小飞 | 191222 | 增加图片存放格式 |
|  |  |  |  |

**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。

# 引言

## 编写目的

为了找到一个简化的架构，更好地运行兼容程序。

## 背景

嵌入式目前有很多运行的操作系统，也有如AUTOSAR架构。目前为了简化这些东西，找到一种可行的方式，把结构进行简化，使功能更加通用；

## 定义

## 参考资料

# 整体设计

## 整体架构



图 2‑1 整体架构

整体架构上来说，只有4个组成部分，BSP、ABI、RTE、CPN。

架构的核心是RTE。通过RTE可以把各个组件链接起来，形成完整的功能。

CPN组件来说没有特别的要求，只要通信接口满足RTE要求就可以在架构之上建立联系，发挥出应有的功能。

## RTE设计

The Run-Time Environment (RTE)运行环境指的就是各个组件运行的基础。其实就是信号的连接。把各个信号连接起来了，组件功能也就发挥出来了。



图 2‑2 RTE结构

RTE信号类型有两类，

1.get/set数据异步传递；

2.trigger/service数据或信号同步传递；

所有的信号都可以按以上两类进行划分，RTE可以统一成以这两类信号进行传输；如果有存在差异的地方也一定可以转成这两类信号。

### set/get接口设计

对于set/get功能来说，set只是把数据存在某个地方，而get是从某个地方进行读取。并不需要特殊的功能，只要能够确保信号能够传输就可以。

对于set来说，把数据放到某个地方，那么这个地方是唯一的，否则就找不到位置存储数据；对于get来说，多个对象可以从同一个位置读取数据。

对于set/get映射关系来说set:get=1:n，n可以为0到无穷大。



图 2‑3 set/get映射关系

对于当前设计来说，set是把数据放到数组某个位置中，get是从这个位置读取。

### set/get接口配置

为了方便于实际应用，当前对于set/get设置了一些宏方便开发。



对于宏使用来说，有以下几个需要进行说明：

|  |  |  |
| --- | --- | --- |
| 名称 | 中文 | 功能 |
| SIGNAL\_SET\_LIST | 发送信号列表 | 建立发送信号与各信号长度 |
| SIGNAL\_GET\_MAP\_LIST | 接收信号映射列表 | 建立接收信号与发送信号关系 |
| SIGNAL\_SETGET\_MAP | 收发信号映射 | 建立收发信号各自映射关系 |
| BUFF\_SETGET\_DEFINE | 收发信号缓存定义 | 建立收发信号各自缓存空间 |
| SIGNAL\_SET\_REGISTER | 发送信号注册名称 | 定义发送信号注册名称 |
| SIGNAL\_GET\_REGISTER | 接收信号注册名称 | 定义接收信号注册名称 |
| SIGNALS\_GETSET\_REGISTER | 信号注册 | 对收发信号按名称进行注册 |

配置步骤：

a.建立发送信号列表

发送信号名称+信号长度（byte）

// 建立发送信号列表

#define SIGNAL\_SET\_LIST(\_)                \

    \_(cpn1\_send1,1)                       \

    \_(cpn1\_send2,4)                       \

    \_(cpn1\_send3,3)                       \

按名称+长度的形式定义宏列表，定义宏可以根据需求去改写，最好可以以SIGNAL\_SET\_LIST作为开头再加上信号类名称，如SIGNAL\_SET\_LIST\_RTEASYN。

b.建立接收信号映射列表

接收信号名称+发送信号名称

// 建立接收信号列表进行映射，配置信号时需要检查与发送信号的长度，默认一致，不符时需要进行报错

#define SIGNAL\_GET\_MAP\_LIST(\_)            \

    \_(cpn2\_receive1,cpn1\_send2)           \

    \_(cpn2\_receive2,cpn1\_send1)           \

    \_(cpn2\_receive3,cpn1\_send3)           \

按接收信号+发送信号的形式定义宏列表。暂时没有检测发送信号和接收信号长度不一致的机制，只能靠人去识别。

c.建立发送信号和接收信号注册名称

分别对已定义的发送信号注册和接收信号注册加上一个名称定义一个新的宏

// 定义需要注册的收发信号

#define ASYN\_SET\_SIGNAL\_REGISTER(signal\_set,len)                SIGNAL\_SET\_REGISTER(RteAsyn,signal\_set,len)

#define ASYN\_GET\_SIGNAL\_REGISTER(signal\_get,signal\_set)         SIGNAL\_GET\_REGISTER(RteAsyn,signal\_get,signal\_set)

d.建立收发映射关系

注册名称+发送信号列表+接收信号列表，三个参数使用对应宏对关系进行映射

// 建立收发信号各自映射关系

SIGNAL\_SETGET\_MAP(RteAsyn,SIGNAL\_SET\_LIST,SIGNAL\_GET\_MAP\_LIST);

e.建立收发缓存

注册名称+发送信号列表+接收信号列表，三个参数使用对应宏对缓存进行定义

// 定义收发信号缓存

BUFF\_SETGET\_DEFINE(RteAsyn,SIGNAL\_SET\_LIST,SIGNAL\_GET\_MAP\_LIST);

f.收发信号进行注册

发送列表+发送信号注册名称

接收列表+接收信号注册名称

使用相应宏对信号进行注册，使发送信号自身关系对应起来，接收信号与发送信号关系对应起来

    // 对信号进行注册

    SIGNALS\_GETSET\_REGISTER(SIGNAL\_SET\_LIST,ASYN\_SET\_SIGNAL\_REGISTER);

    SIGNALS\_GETSET\_REGISTER(SIGNAL\_GET\_MAP\_LIST,ASYN\_GET\_SIGNAL\_REGISTER);

### serv/trig接口设计

对于serv/trig接口相对于彼此来说，关系就只是直接调用。通过RTE进行转接是因为可以把所有接收做成通用的接口，不再识别内容，只要完成相应的信号配对就可以实现信号传输。

对于trig/serv的映射关系来说，trig:serv=n:1，n为0到无穷大。



图 2‑4 trig/serv映射关系

RTE对于trig/serv的作用是抹平接口对接关系，只要识别信号就可以完成相应的动作过程。

### serv/trig接口配置

同样对于serv/trig来说也设置了相应宏，方便开发过程中的处理；



对于宏使用来说，有以下几个需要进行说明：

|  |  |  |
| --- | --- | --- |
| 名称 | 中文 | 功能 |
| SIGNAL\_SERV\_LIST | 服务信号列表 | 建立服务信号与各信号长度 |
| SIGNAL\_TRIG\_MAP\_LIST | 触发信号映射列表 | 建立触发信号与发送信号关系 |
| SIGNAL\_SERVTRIG\_MAP | 服务触发信号映射 | 建立服务触发信号各自映射关系 |
| BUFF\_SERVTRIG\_DEFINE | 服务触发信号缓存定义 | 建立服务触发信号各自缓存空间 |
| SIGNAL\_SERV\_REGISTER | 服务信号注册名称 | 定义服务信号注册名称 |
| SIGNAL\_TRIG\_REGISTER | 触发信号注册名称 | 定义触发信号注册名称 |
| SIGNALS\_SERVTRIG\_REGISTER | 信号注册 | 对服务触发信号按名称进行注册 |

配置步骤：

a.建立服务信号列表

服务信号名称+服务函数+参数大小（byte）

// 建立服务信号列表

// 服务信号一定要建立与触发信号一致的传输类型数据类型，用于识别大小，而不依赖触发信号

#define SIGNAL\_SERV\_LIST(\_)                                           \

    \_(cpn1\_serv1,cpn1\_serv1\_serv,0)                                   \

    \_(cpn1\_serv2,cpn1\_serv2\_serv,sizeof(ulServ2Unpack))               \

    \_(cpn1\_serv3,cpn1\_serv3\_serv,sizeof(ucServ3Buff))                 \

按名称+服务函数+参数长度的形式定义宏列表，定义宏可以根据需求去改写，最好可以以SIGNAL\_SERV\_LIST作为开头再加上信号类名称，如SIGNAL\_SERV\_LIST \_RTEASYN。

服务函数一定得要预先知道，否则列表无法建立。对于服务函数没参数的，需要把参数大小设置为0。

b.建立触发信号映射列表

触发信号名称+服务信号名称+参数大小（byte）

// 建立触发信号列表进行映射，配置信号时需要检查与服务信号的长度，默认一致，不符时需要进行报错

// 触发信号类型定义可以独立于服务数据类型，只要保证大小一致

#define SIGNAL\_TRIG\_MAP\_LIST(\_)                                       \

    \_(cpn2\_trig1,cpn1\_serv2,sizeof(ulServ2Unpack))                    \

    \_(cpn2\_trig2,cpn1\_serv3,sizeof(ucServ3Buff))                      \

    \_(cpn2\_trig3,cpn1\_serv1,0)                                        \

按触发信号名称+服务信号名称+参数大小的形式定义宏列表。触发信号大小可以不使用服务信号参数作为触发信号大小填写，但写触发参数大小时要与服务信号参数大小一致。

c.建立服务信号和触发信号注册名称

分别对已定义的服务信号注册和触发信号注册加上一个名称定义一个新的宏

// 定义需要注册的服务触发信号

#define SYN\_SERV\_SIGNAL\_REGISTER(signal\_serv,serv,len)                 SIGNAL\_SERV\_REGISTER(RteSyn,signal\_serv,serv,len)

#define SYN\_TRIG\_SIGNAL\_REGISTER(signal\_trig,signal\_serv,len)          SIGNAL\_TRIG\_REGISTER(RteSyn,signal\_trig,signal\_serv,len)

d.建立服务触发映射关系

注册名称+服务信号列表+触发信号列表，三个参数使用对应宏对关系进行映射

// 建立服务触发信号映射

SIGNAL\_SERVTRIG\_MAP(RteSyn,SIGNAL\_SERV\_LIST,SIGNAL\_TRIG\_MAP\_LIST);

e.建立服务触发缓存

注册名称+服务信号列表+触发信号列表，三个参数使用对应宏对缓存进行定义

// 定义服务触发信号缓存

BUFF\_SERVTRIG\_DEFINE(RteSyn,SIGNAL\_SERV\_LIST,SIGNAL\_TRIG\_MAP\_LIST);

f.服务触发信号进行注册

服务列表+服务信号注册名称

触发列表+触发信号注册名称

使用相应宏对信号进行注册，使服务信号自身关系对应起来，触发信号与服务信号关系对应起来

    // 对信号进行注册

    SIGNALS\_SERVTRIG\_REGISTER(SIGNAL\_SERV\_LIST,SYN\_SERV\_SIGNAL\_REGISTER);

    SIGNALS\_SERVTRIG\_REGISTER(SIGNAL\_TRIG\_MAP\_LIST,SYN\_TRIG\_SIGNAL\_REGISTER);

RTE完整配置工程如下：



## CPN组件设计

component（CPN）组件，属于应用结构。不用的应用可以定制不同的功能，标准化，模块化之后可以形成很稳定的功能。

对于应用组件来说有一个比较特殊的是，调度组件。调度组件可以决定其他组件功能能否正常实施。调度组件可以说是组织架构的一部分。

### 组件信号配置

对于整体架构来说，信号就分为两类。set/get和serv/trig两类，并且两类信号分别定义了操作类。RteAsyn类和RteSyn类。

#### RTE信号

对于信号操作来说，目前定义的两个操作类RteAsyn和RteSyn分别只实例化了一个。亦即所有的信号必须在这两个实例中进行配置。

对于实际应用来说这两个类不是只能实例化两个实例，但就目前应用来说，两个类已经够用了。就规划来说，先这么用着，后续有需求再进行扩充。

##### RTE信号配置

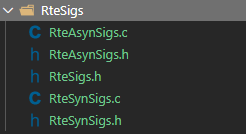


图 2‑5 RTE信号结构

RTE信号就只是对同步异步信号进行实例化。

###### RteSigs.h

信号文件主要是把当前需要添加的组件添加进来。

/\*头文件包含\*/

#include "../Rte/Rte.h"

/\*组件信号\*/

#include "../Abi/AbiSigs.h"

#include "../CpnSch/CpnSchSigs.h"

#include "../CpnSigs/Cpn1.h"

#include "../CpnSigs/Cpn2.h"

#include "../CpnSigs/Cpn3.h"

#include "../CpnSigs/Cpn4.h"

/\*所有信号进行匹配\*/

#include "./RteAsynSigs.h"

#include "./RteSynSigs.h"

在组件信号中把需要添加的组件包含进来。

###### RteAsynSigs.h

在异步信号文件中只对需要添加的组件信号列表进行添加。

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

\* get set信号列表

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

/\*发送信号列表

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

#define SIGNAL\_SET\_LIST\_RTEASYN(\_)                         \

        SIGNAL\_SET\_LIST\_RTEASYN\_CPN1(\_)                    \

        SIGNAL\_SET\_LIST\_RTEASYN\_CPN3(\_)                    \

/\*接收信号列表

\* 配置信号时需要检查发送信号的长度对信号进行映射

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

#define SIGNAL\_GET\_MAP\_LIST\_RTEASYN(\_)                     \

        SIGNAL\_GET\_MAP\_LIST\_RTEASYN\_CPN2(\_)                \

        SIGNAL\_GET\_MAP\_LIST\_RTEASYN\_CPN4(\_)                \

比如需要添加SIGNAL\_SET\_LIST\_RTEASYN\_CPN1(\_)，在SIGNAL\_SET\_LIST\_RTEASYN(\_)对应的列表上添加SIGNAL\_SET\_LIST\_RTEASYN\_CPN1(\_)就可以了，不需要管顺序，但必须要有。

接收列表也是同样的添加方法。

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

\* 注册get set信号

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

/\*注册发送信号

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

#define ASYN\_SET\_SIGNAL\_REGISTER(signal\_set,len)                SIGNAL\_SET\_REGISTER(RteAsynSigs,signal\_set,len)

/\*注册接收信号

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

#define ASYN\_GET\_SIGNAL\_REGISTER(signal\_get,signal\_set)         SIGNAL\_GET\_REGISTER(RteAsynSigs,signal\_get,signal\_set)

注册收发信号，已根据总的列表进行添加，已不需要更改。

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

\* get set信号功能模块

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

/\*建立收发信号各自映射关系

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

SIGNAL\_SETGET\_MAP(RteAsynSigs,SIGNAL\_SET\_LIST\_RTEASYN,SIGNAL\_GET\_MAP\_LIST\_RTEASYN);

/\*异步信号初始化函数

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

void RteAsynSetGetInit(void);

/\*定义收发类

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

extern RteAsyn clRteAsynSigs;

信号映射、异步信号初始化以及异步信号实例化类声明，已定义好不需要更改，特别是信号映射，会根据总的信号列表进行匹配定义。

###### RteAsynSigs.c

/\*头文件包含\*/

#include "./RteAsynSigs.h"

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

\* set get定义

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

/\*定义收发信号缓存

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

BUFF\_SETGET\_DEFINE(RteAsynSigs,SIGNAL\_SET\_LIST\_RTEASYN,SIGNAL\_GET\_MAP\_LIST\_RTEASYN);

/\*定义收发类

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

RteAsyn clRteAsynSigs;

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

\* get set初始化

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

/\*注册信号及初始化发送接收类

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

void RteAsynSetGetInit(void)

{

    /\*对信号进行注册

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

    SIGNALS\_GETSET\_REGISTER(SIGNAL\_SET\_LIST\_RTEASYN,ASYN\_SET\_SIGNAL\_REGISTER);

    SIGNALS\_GETSET\_REGISTER(SIGNAL\_GET\_MAP\_LIST\_RTEASYN,ASYN\_GET\_SIGNAL\_REGISTER);

    /\*初始化调用类

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

    CN(RteAsyn,&clRteAsynSigs,RteAsynSigsSigSetBuff,RteAsynSigsSigSetIndex,RteAsynSigsSigGetIndex);

    if(OPRS(clRteAsynSigs) != NULL){}

}

异步信号实例化处理，以及异步信号缓存待会根据收发信号列表进行配置。本文件内容都不需要改动，会自动匹配生成。

###### RteSynSigs.h

在同步信号文件中只对需要添加的组件信号列表进行添加。

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

\* trig serv信号列表

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

/\*服务信号列表

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

#define SIGNAL\_SERV\_LIST\_RTESYN(\_)                         \

        SIGNAL\_SERV\_LIST\_RTESYN\_ABI(\_)                     \

        SIGNAL\_SERV\_LIST\_RTESYN\_CPNSCH(\_)                  \

        SIGNAL\_SERV\_LIST\_RTESYN\_CPN1(\_)                    \

        SIGNAL\_SERV\_LIST\_RTESYN\_CPN2(\_)                    \

        SIGNAL\_SERV\_LIST\_RTESYN\_CPN3(\_)                    \

        SIGNAL\_SERV\_LIST\_RTESYN\_CPN4(\_)                    \

/\*触发信号列表

\* 配置信号时需要检查服务信号的长度对信号进行映射

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

#define SIGNAL\_TRIG\_MAP\_LIST\_RTESYN(\_)                     \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_ABI(\_)                 \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPNSCH(\_)              \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPN1(\_)                \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPN2(\_)                \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPN3(\_)                \

        SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPN4(\_)                \

比如需要添加SIGNAL\_SERV\_LIST\_RTESYN\_CPNSCH(\_)，在SIGNAL\_SERV\_LIST\_RTESYN(\_)对应的列表上添加SIGNAL\_SERV\_LIST\_RTESYN\_CPNSCH(\_)就可以了，不需要管顺序，但必须要有。

触发列表也是同样的添加方法。

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

\* 注册trig serv信号

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

/\*注册服务信号

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

#define SYN\_SERV\_SIGNAL\_REGISTER(signal\_serv,serv)                 SIGNAL\_SERV\_REGISTER(RteSynSigs,signal\_serv,serv)

/\*注册触发信号

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

#define SYN\_TRIG\_SIGNAL\_REGISTER(signal\_trig,signal\_serv)          SIGNAL\_TRIG\_REGISTER(RteSynSigs,signal\_trig,signal\_serv)

注册服务触发信号，已根据总的列表进行添加，已不需要更改。

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

\* trig serv信号功能模块

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

/\*建立服务触发信号映射

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

SIGNAL\_SERVTRIG\_MAP(RteSynSigs,SIGNAL\_SERV\_LIST\_RTESYN,SIGNAL\_TRIG\_MAP\_LIST\_RTESYN);

/\*同步信号初始化函数

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

void RetSynServTrigInit(void);

/\*定义服务触发类

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

extern RteSyn clRteSynSigs;

信号映射、同步信号初始化以及同步信号实例化类声明，已定义好不需要更改，特别是信号映射，会根据总的信号列表进行匹配定义。

###### RteSynSigs.c

/\*头文件包含\*/

#include "./RteSynSigs.h"

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

\* trig serv定义

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

/\*定义服务触发信号缓存

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

BUFF\_SERVTRIG\_DEFINE(RteSynSigs,SIGNAL\_SERV\_LIST\_RTESYN,SIGNAL\_TRIG\_MAP\_LIST\_RTESYN);

/\*定义服务触发类

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

RteSyn clRteSynSigs;

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

\* trig serv初始化

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

/\*注册信号及初始化服务触发类

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

void RetSynServTrigInit(void)

{

    /\*对信号进行注册

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

    SIGNALS\_SERVTRIG\_REGISTER(SIGNAL\_SERV\_LIST\_RTESYN,SYN\_SERV\_SIGNAL\_REGISTER);

    SIGNALS\_SERVTRIG\_REGISTER(SIGNAL\_TRIG\_MAP\_LIST\_RTESYN,SYN\_TRIG\_SIGNAL\_REGISTER);

    /\*初始化调用类

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

    CN(RteSyn,&clRteSynSigs,RteSynSigsSigServBuff,RteSynSigsSigTrigIndex);

    if(OPRS(clRteSynSigs) != NULL){}

}

同步信号实例化处理，以及同步信号缓存待会根据收发信号列表进行配置。本文件内容都不需要改动，会自动匹配生成。

#### 组件信号

对于组件来说，分为两部分，一是组件类，二是组件信号。

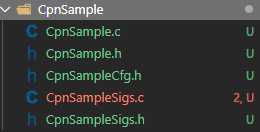


图 2‑6 组件结构

对于组件类来说就是一个普通的C类。添加对应的方法就行了。

##### CpnSampleSig.h

对于组件信号来说需要定义以下内容：

发送数据类型，

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

\* get set 数据类型定义

\* 一般只定义set数据，get类型引用当前头文件

\* 只需要把发送的数据定义出来，收发接口使用的是统一接口，无其他定义

\* 信号数据类型一般来说都得强制定义一遍，否则很难知道信号对应数据是什么

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

/\*发送数据类型

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

发送数据类型需要去定义的。即使是已常用的数据类型使用在某个信号上时，需要根据信号去定义一个名称。推荐去定义类型，如果真不想去定义专用类型，那么可在信号名称上标示类型，如sig\_set\_Cpn1\_send1\_uint16。

服务数据类型，

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

\* serv trig 数据类型定义

\* 一般只定义serv数据，trig类型引用当前头文件

\* 只需要把发送的数据定义出来，收发接口使用的是统一接口，无其他定义

\* 信号数据类型一般来说都得强制定义一遍，否则很难知道信号对应数据是什么

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

/\*服务数据类型

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

与发送数据类型一样，最好有专用数据类型定义，否则最好在信号名称进行标示。

服务函数，触发接口，

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

\* serv trig 信号接口定义

\* 类继承的接口就是触发信号接口，在接口函数中使用触发信号进行触发

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

/\*组件初始化服务

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

void serv\_CpnSample\_init(void);

/\*服务

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

void serv\_CpnSample\_run(void);

/\*服务

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

void serv\_CpnSample\_update(void);

/\*服务

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

void serv\_CpnSample\_excute(void);

/\*接口触发

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

//void trig\_CpnSample\_inf\_infFunc(void);

对于组件来说，这是组件与外部进行关联的重要接口。服务提供被调用功能。接口触发对信号进行其他组件信号触发。

收发列表，

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

\* get set信号列表

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

/\*发送信号列表

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

#define SIGNAL\_SET\_LIST\_RTEASYN\_CPNSAMPLE(\_)                \

/\*接收信号列表

\* 配置信号时需要检查发送信号的长度对信号进行映射

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

#define SIGNAL\_GET\_MAP\_LIST\_RTEASYN\_CPNSAMPLE(\_)            \

收发信号不会和接口函数进行关联，只需根据需求定义相关信号即可。

发送信号定义规则如下，

sig\_set\_CpnSample\_send1\_uint16

信号头是固定的，组件名称因组件而固定，信号名称根据需求进行选取，最后加上信号传递数据类型。

接收信号定义规则如下，

sig\_get\_CpnSample\_receive1\_uint16

信号头是固定的，组件名称因组件而固定，信号名称根据需求进行选取，最后加上信号传递数据类型。

服务触发列表，

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

\* trig serv信号列表

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

/\*服务信号列表

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

#define SIGNAL\_SERV\_LIST\_RTESYN\_CPNSAMPLE(\_)                \

    \_(sig\_serv\_CpnSample\_init,serv\_CpnSample\_init)          \

    \_(sig\_serv\_CpnSample\_run,serv\_CpnSample\_run)            \

    \_(sig\_serv\_CpnSample\_update,serv\_CpnSample\_update)      \

    \_(sig\_serv\_CpnSample\_excute,serv\_CpnSample\_excute)      \

/\*触发信号列表

\* 配置信号时需要检查服务信号的长度对信号进行映射

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

#define SIGNAL\_TRIG\_MAP\_LIST\_RTESYNCPNSAMPLE(\_)             \

服务信号需要和函数进行关联。服务信号很多时候也会有一些数据的传递，但为了统一，信号名称上还是不加数据类型。

服务信号定义规则如下，

sig\_serv\_CpnSample\_serv1

信号头是固定的，组件名称因组件而固定，信号名称根据需求进行选取。

触发信号定义规则如下，

sig\_trig\_CpnSample\_trig1

信号头是固定的，组件名称因组件而固定，信号名称根据需求进行选取。

组件任务，

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

\* 组件运行任务

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

/\*运行任务声明

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

void taskCpnSample(void);

组件任务进行声明，此任务是一个相对特殊的函数，在调度组件CpnSch中会时行调用，在初始化服务中会把任务传递出去。

组件任务命名规则，

taskCpnSample

task为固定头名称，后加组件名称组成任务名称。

##### CpnSampleSig.c

组件类实例化，

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

\* CpnSample类定义

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

/\*CpnSample类

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

CpnSample clCpnSample;

功能固定，对组件类进行实例化，组件实例也是固定名称clCpnSample。cl为实例固定名称。

服务触发函数，

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

\* serv trig 信号接口定义

\* 类继承的接口就是触发信号接口，在接口函数中使用触发信号进行触发

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

/\*组件初始化服务

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

void serv\_CpnSample\_init(void){

    CN(CpnSample,&clCpnSample);

    if(OPRS(clCpnSample) != NULL){}

}

/\*服务

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

void serv\_CpnSample\_run(void){}

/\*服务

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

void serv\_CpnSample\_update(void){}

/\*服务

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

void serv\_CpnSample\_excute(void){}

/\*接口触发

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

//void trig\_CpnSample\_inf\_infFunc(void){}

服务触发数据根据需求进行配置。

##### CpnSampleCfg.h

根据需求设置组件配置信息，如不需要可时行删除。

#### 脚本生成

对于一个新组件来说要把整个组件的信号一步一步写一遍，很机械式的操作很费劲，当前使用脚本的形式对信号进行配置。

当前使用脚本进行生成，是根据C类来进行配置的。只要C类的功能函数都会默认配置为服务函数。

脚本使用规则：

|  |  |
| --- | --- |
| 规则 | python cpnTemplate.py path CpnName fun1 fun2 … fun(n) |
| 示例 | python cpnTemplate.py CpnSample CpnSample run excute update |





### CpnSch调度组件

调度组件不是必须，但当前为了在当前架构上有效地运行，进行了一个默认的调度设计。使用其他的调度方式进行也可以，合理组织运行就行。

#### 任务组



图 2‑7 任务组

为了使任务进行高效调度，任务进行分组管理，每个任务组大小可以配置有32个任务，16个任务，8个任务。上图上是32个任务的任务组示例。

起始节拍，记录任务初始运行时刻。在任务初始化时，是在当前tick基础上进行叠加。如果是周期任务，则会在周期执行完成后叠加周期值。

调度节拍，任务调度周期。如果是一次性调度任务此值为0。

任务激活请求表，任务到达执行时间会把对应标志位进行置位，执行完成后对此位进行清除。

未启动表，对应任务未启动标志为1，已启动后标志为0；已启动表示已进入周期执行；

任务分配表，对任务进行初始化时，任务组内此标志对应任务没有被占用时就可以进行置位，初始化对应位置为任务。

任务记录表，任务初始化时存储任务函数。

任务ID优先级，任务ID越低优先执行。ID并没有定义相应参数，只是一个标号进行示意。

#### 任务等级



图 2‑8 任务等级

任务进行管理设置了不同的等级，level0是背景任务，只要没有高等级任务就会运行的；level1是一级任务，level2是二级任务，level3是三级任务。

任务优先级，

level0<level1<level2<level3

说明：

1.level0任务是背景任务，只要没有更高等级的任务组就会一直循环执行；

2.每个任务等级可以定义多个任务组，对于每个等级来说是独立处理等级内部任务组；

3.普通的调度只要level0和level1就够了，实时处理再高一点level2都能满足；

4.只有在一些极端的情况下才可以需要level3，目前也只是作为一个预留；

#### 任务调度结构



图 2‑9 任务调度结构

说明：

1.tick是由定时器直接驱动的一个变量，但定时器不参与到状态机函数的调用，只会作用到tick；

2.背景任务更新只有在所有的背景任务都执行了一遍之后才会进入；即任务激活都为0；

3.任务执行状态一次只执行一个任务，循环查询相关状态；

#### 组件信号

服务触发信号，

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

\* trig serv信号列表

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

/\*服务信号列表

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

#define SIGNAL\_SERV\_LIST\_RTESYN\_CPNSCH(\_)                  \

    \_(sig\_serv\_CpnSch\_addTask,serv\_CpnSch\_addTask)         \

    \_(sig\_serv\_CpnSch\_delTask,serv\_CpnSch\_delTask)         \

    \_(sig\_serv\_CpnSch\_delay,serv\_CpnSch\_delay)             \

    \_(sig\_serv\_CpnSch\_now,serv\_CpnSch\_now)                 \

    \_(sig\_serv\_CpnSch\_init,serv\_CpnSch\_init)               \

    \_(sig\_serv\_CpnSch\_run,serv\_CpnSch\_run)                 \

    \_(sig\_serv\_CpnSch\_tick,serv\_CpnSch\_tick)               \

/\*触发信号列表

\* 配置信号时需要检查服务信号的长度对信号进行映射

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

#define SIGNAL\_TRIG\_MAP\_LIST\_RTESYN\_CPNSCH(\_)              \

    \_(sig\_trig\_CpnSch\_inf\_getTime,sig\_serv\_Abi\_getTime)    \

    \_(sig\_trig\_CpnSch\_inf\_setErr,sig\_serv\_Abi\_setErr)      \

    \_(sig\_trig\_Cpn1\_taskInit,sig\_serv\_Cpn1\_taskInit)       \

    \_(sig\_trig\_Cpn2\_taskInit,sig\_serv\_Cpn2\_taskInit)       \

    \_(sig\_trig\_Cpn3\_taskInit,sig\_serv\_Cpn3\_taskInit)       \

    \_(sig\_trig\_Cpn4\_taskInit,sig\_serv\_Cpn4\_taskInit)       \

调度组件提供以下服务，

添加任务，删除任务，延时，精确时间的功能。

初始化，运行，tick驱动也是相关服务，由ABI进行触发驱动。

调度组件提供以下触发，

触发ABI获取精确时间。

触发CpnSch内部产生错误发送到总线。

触发其他组件的初始化。其他组件只要需要使用CpnSch进行调度就需要在触发信号上进行映射，然后在初始化服务中对信号进行触发。

#### 组件参数配置

// 任务功能参数配置

typedef uint16 taskGroupType;                                                                // 任务组类型 可选 uint8 uint16 uint32

#define CPN\_SCH\_TASK\_MEASURE\_ENABLE                  TRUE                                    // 任务运行时间测量 TRUE FALSE

#define CPN\_SCH\_TASK\_TICK\_TIME\_US                    500U                                    // 单步时间500US

// 每个任务组可以存放sizeof(taskGroupType)\*8个任务，每个任务等级可以配置不同任务组数

#define CPN\_SCH\_LEVEL0\_GROUP\_NUM\_CFG                 2U                                      // level0（背景）任务组数配置

#define CPN\_SCH\_LEVEL1\_GROUP\_NUM\_CFG                 2U                                      // level1任务组数配置

#define CPN\_SCH\_LEVEL2\_GROUP\_NUM\_CFG                 1U                                      // level2任务组数配置

#define CPN\_SCH\_LEVEL3\_GROUP\_NUM\_CFG                 0U                                      // level3任务组数配置

#define CPN\_SCH\_GROUP\_TOTAL\_NUMS                     (CPN\_SCH\_LEVEL0\_GROUP\_NUM\_CFG  \

                                                     + CPN\_SCH\_LEVEL1\_GROUP\_NUM\_CFG \

                                                     + CPN\_SCH\_LEVEL2\_GROUP\_NUM\_CFG \

                                                     + CPN\_SCH\_LEVEL3\_GROUP\_NUM\_CFG)         // 总任务组数

tick步长时间需要事先定义好，和底层定义实现有关系，比较推荐的时间是200ms，500ms。其他时间也是可以，看系统灵活性和需求。

不同任务等级对应的任务组数是可以进行配置的，如果某个任务组不需要可以把参数设置为0。level0和level1是必须的，这是调度的最小系统。

### CpnErr错误处理组件

错误处理组件也不是必须的。但当前架构下，设计了这样一个组件，用于在组件调度时产生的错误处理。

当前组件只进行了错误个数的统计，后续功能待完善。

## ABI设计

Abstract Interface,根据上层组件需求进行信号的抽象，主要作用是隔离BSP层。上层组件信号只反馈到ABI，ABI相当于给出一个标准的信号接口。与BSP也只是完成信号对接就可以了。

上层组件需要底层信号就在ABI定义就可以了。

有可能对一些组件进行测试，类似做一些桩的功能，也可以直接在ABI层进行定义。这样的速度最快，实现的功能也最简单。

## BSP结构

Board Support Package板级支持包，无特殊定义。只要满足ABI层对接，可以使ABI层完成底层信号的抽象。

# 组件设计

**当前组件的脚本也很完善了，只需要关注组件间的信号对接就可以。组件的添加、删除、新建功能很简单了。**

## 脚本说明

当前控制脚本主要有以下几个：

C模板、OOPC对象、SM状态机、CPN新建、CPN操作等。

### C模板

使用方法：python cTemplate.py path filename

脚本只能有3个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

filename根据需要选取

### OOPC对象

脚本本身的代码量并不大，使用规则也很简单。Python一般都是指令的方式。

当前脚本指令为: python classTemplate.py path className func\_list

脚本至少需要4个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

类名称className根据需要选取

func\_list是多个参数，目前配置类时至少需要给定1个功能函数

示例，

脚本参数：

python classTemplate.py mypath/path My func second third

会在mypath/path路径下生成名为My的类，除去init功能函数，还有func/second/third三个功能函数。

### SM状态机

当前脚本指令为:python sm.py path stateMachineName stateList

脚本至少需要4个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

状态机名称stateMachineName根据需要选取

stateList是多个参数，目前配置状态机时至少需要给定1个状态

示例，

脚本参数：

python sm.py mypath mysm examp next

会在mypath下生成名为mysm的状态机，除去init/default两个状态，还有examp/next两个状态。

### CPN新建

使用方法：python cpnTemplate.py path cpnName func\_list

脚本至少需要4个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

cpnName根据需要选取

func\_list是多个参数，目前配置类时至少需要给定1个功能函数

功能函数也分为oopc功能函数和serv信号功能函数，serv信号功能函数必须以/或\开头

功能函数在编写OOPC功能函数和serv信号功能函数时可以混着写，只要serv信号功能函数前方带上/或\就可以。

示例，

脚本参数：

python cpnTemplate.py CpnApps/CpnNew CpnNew examp /get /set next

会在CpnApps/CpnNew下生成名为CpnNew的组件（包含CpnNew类和CpnNew信号）。

CpnNew类除去init功能函数，还有example/next二个功能函数。

CpnNew信号包含get/set两个服务信号。

### CPN操作

使用方法：python cpnOperation.py path cpnName operation

脚本至少需要4个参数，

路径path，当前路径可以使用/或\，其他路径一定要以非/或\开头，否则生成不了

cpnName根据需要选取

operation，可以是add,del操作

当前操作函数只有增删，操作函数可以同时有多个是作为一个列表进行处理的。目前只有两个操作也可以连起来操作，比如先del,再add，可以把对应的组件调整到最后。

示例，

脚本参数：

python cpnOperation.py CpnApps/CpnNew CpnNew del

会把CpnNew组件在目前的调度系统中删除。然后整理一下组件信号的连接关系，最后只要CpnApps/CpnNew对应的组件删除工作重新编译就可以重新工作了。

## 组件组成

组件由两部分内容组成，信号和组件对象。

### 组件信号

组件信号收RteSigsConfigList.h进行的映射配置，只需要把组件相应的信号添加进去就可以自动映射。

信号内容查看RTE设计说明。

### 组件对象

对于组件功能的实现，目前是使用面向对象来进行管理的。面向对象可以找前期编写的OOPC相关说明。

## 组件匹配

对于应用设计来说，最应该关注的就是组件间信号的匹配。

组件的添加与删除都可以通过脚本进行控制，但对于具体应用来说，信号的连接是根据需求而进行确定。

功能完整的组件，信号配置正确之后就可以正常运行。