**runstuct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**

[**runstuct运行架构** I](file:///D:\MyFiles\github\runStuct\doc\runStuct运行架构_191222.docx#_Toc27925328)

[**目录** ii](#_Toc27925329)

[**修订记录：** iii](#_Toc27925330)

[**文档说明：** iii](#_Toc27925331)

[1. 引言 - 1 -](#_Toc27925332)

[1.1. 编写目的 - 1 -](#_Toc27925333)

[1.2. 背景 - 1 -](#_Toc27925334)

[1.3. 定义 - 1 -](#_Toc27925335)

[1.4. 参考资料 - 1 -](#_Toc27925336)

[2. 整体设计 - 2 -](#_Toc27925337)

[2.1. 整体架构 - 2 -](#_Toc27925338)

[2.2. RTE设计 - 2 -](#_Toc27925339)

[2.2.1. set/get接口设计 - 3 -](#_Toc27925340)

[2.2.2. set/get接口配置 - 4 -](#_Toc27925341)

[2.2.3. serv/trig接口设计 - 6 -](#_Toc27925342)

[2.2.4. serv/trig接口配置 - 6 -](#_Toc27925343)

[2.3. CPN组件设计 - 9 -](#_Toc27925344)

[2.3.1. cpnSch调度组件 - 9 -](#_Toc27925345)

[2.4. ABI设计 - 9 -](#_Toc27925346)

[2.5. BSP结构 - 10 -](#_Toc27925347)

[3. 组件设计 - 11 -](#_Toc27925348)

[3.1. 功能模块1 - 11 -](#_Toc27925349)

[3.1.1. 具体说明1 - 11 -](#_Toc27925350)

[3.1.2. 具体说明2 - 11 -](#_Toc27925351)

[3.2. 功能模块2 - 11 -](#_Toc27925352)

[3.2.1. 具体说明1 - 11 -](#_Toc27925353)

[3.2.2. 具体说明2 - 11 -](#_Toc27925354)

**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。

### cpnSch调度组件

## ABI设计

## BSP结构

# 组件设计

## 功能模块1

### 具体说明1

### 具体说明2

## 功能模块2

### 具体说明1

### 具体说明2