INFY POWER
英飞源技术

PDU_CORE 功率分配静态库应用说明

Innovation For Your Power

深圳英飞源技术有限公司

修改记录

版本	修改内容	修改日期
1.01	测试节点固定路径，添加 demo 验证程序	2025.09.10

方案描述

■ 本方案作为 PCU 软件的一个中间件，采用**模块化设计**，确保系统具备高可靠性、可扩展性和可维护性。系统设计遵循**高内聚低耦合**原则，各个模块职责清晰，通过定义接口进行通信。方案能够智能分配充电功率，支持多枪并行充电，根据车辆需求动态调整功率分配，并确保系统在各种工况下的安全运行。

■本方案通过配置功率生成的拓扑描述，将功率节点间的电气连接和充电桩接入功率池的电气连接内化为内核式链表，确保软件稳定可靠地检索功率节点。

■在分配策略业务上采用**有限状态机**与**事件驱动**结合的方式，根据不同的分配策略调整功率节点选择逻辑和功率设置参数。

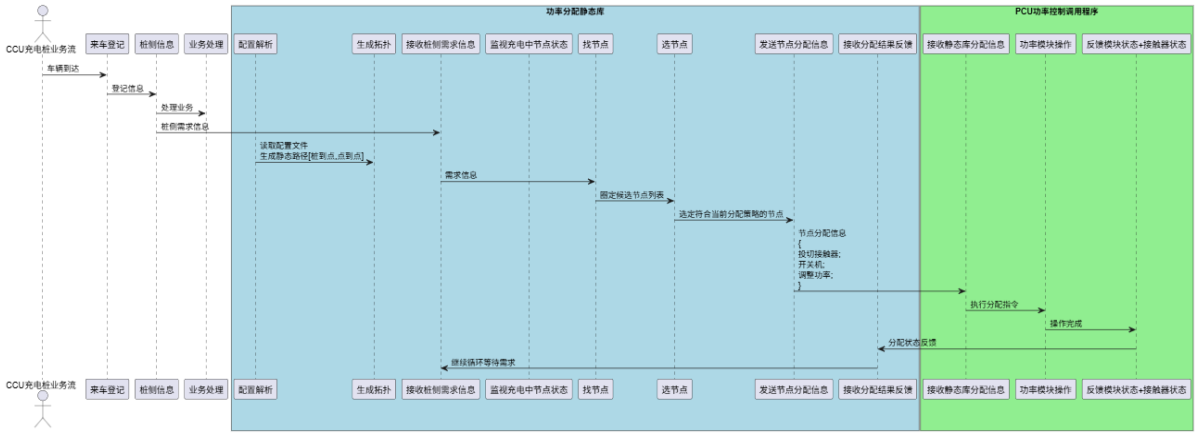

图 1

数据结构

■功率节点间的电气连接及充电桩可调用功率池个数及连接功率池接触器的序号虽然是非固定的，但在较长的运行时间范围内是不变的，所以软件初始化时应该有一个考虑可扩充枪位、柜位并满足最大运行模式的配置，如果在运行中有电气结构增减可以通过 upper 修改参数设置实现**可在应用配置。**

■本方案中通过配置工具生成拓扑结构基本信息，在此基础上由 upper 根据现场应用场景配置关键信息，拓扑结构由两种方式搭配共同完成；

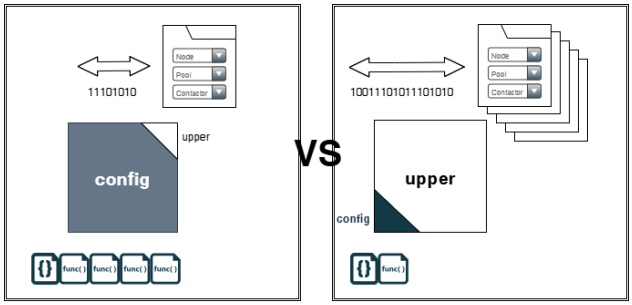

图 2

■拓扑结构在软件上体现为对数据单元的内核式链表互联；功率节点之间以及充电桩枪头与功率节点（模块）之间的物理电气连接，被抽象为数据单元之间的链表结构；每个设备对应一个数据对象，对象之间通过指针或引用形成链式关系，从而反映实际的电气拓扑

■软件对数据维护的日常操作即：

- 通过链表节点侵入式访问数据结构内容（如由一个功率节点获取当前节点的额定功率）
- 通过链表节点前推回溯其他节点（如通过枪头寻找下一个符合分配条件的功率节点）
- 通过链表节遍历追溯链表头（如从一个功率节点找到与该节点连接的充电桩）
- 通过链表头遍历所有链表节点（如根据枪头找到所有与其电气连接的功率节点）
- 对链表进行节点插入和解除（如响应 upper 下发设置重构电气拓扑）

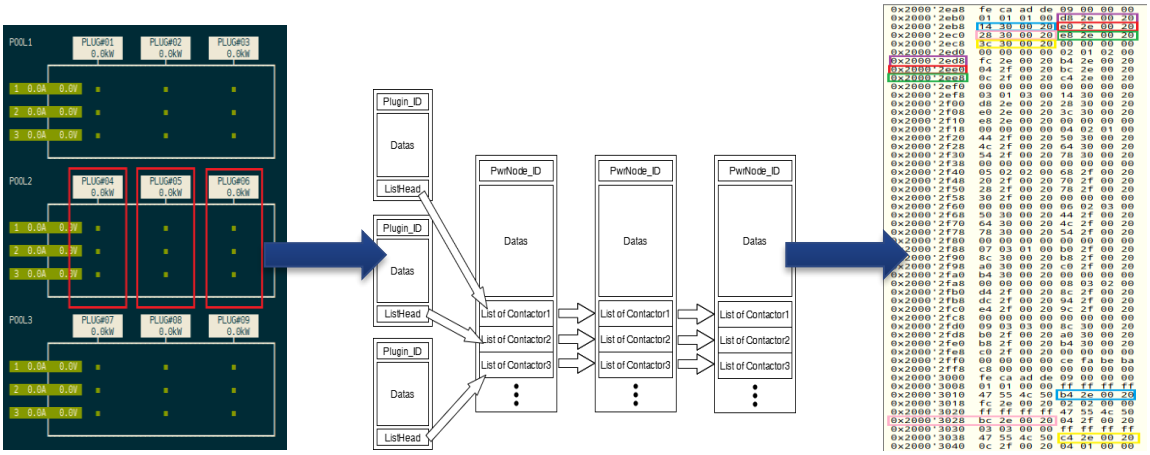

图 3

■枪头的充电操作在软件中体现为通过普通链表对功率节点的互联

注意充电链表是相对描述电气拓扑关系的内核式链表更简单快捷的另一种链表

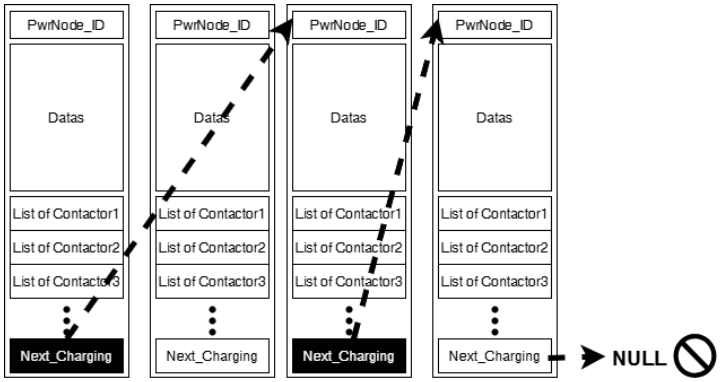

图 4

枪头通过该链表实现对功率节点的并解列操作，包括如下：

- 通过插入链表节点实现对功率节点并列
- 通过解除链表节点实现对功率节点解列
- 通过遍历链表实现对在充功率节点的查询，从而进行下一步逻辑决策

软件配置

■以上所论述可反应实物电气拓扑的数据结构可以通过两个软件方法写入

- 编译时过配置工具生成头文件进行静态构造
- 运行时通过 upper 下发配置参数进行动态重构或析构
- 配置工具构造的电气拓扑仅作为最大可能运行模式配置，可定义功率池个数、池内功率节点容量、节点配备接触器数量，以及对枪头跨接枪位进行初步配置
- upper 可以在设备发货后根据现场情况对拓扑结构增、删、改、换，自由重构
- 实际应用中，两种配置途径应用场景不同；配置工具通常应用于产品投运前、电气设计形成初步沟通后的发货烧录工作，并且可以大量节省 upper 配置项；upper 仅根据已有拓扑做针对性关键配置，通过归纳精简后的若干项配置进行重构，而不是重复配置工具的工作

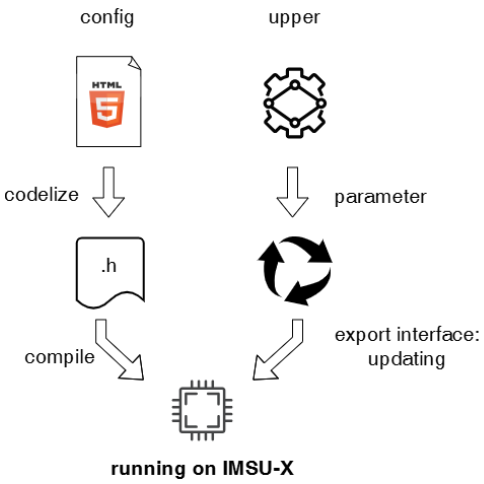

图 5

- 配置工具在设计上考虑兼容线环拓扑结构，对静态库提供统一的设置接口

■配置工具为内置 Javascript 的 HTML 文件，通过浏览器页面执行①【下拉框勾选】→②【拓扑图检视】→③【生成配置文件】的操作序列，将生成的头文件与代码一起编译

图 6

⚠配置工具无法参与程序运行后的配置，仅改变编译器的编译行为，拓扑结构描述置于烧写 BIN 文件中

拓扑配置工具在本方案中并不是一个可视化工具，而是作为 IMSU-X 嵌入式**准低代码开发**（Semi Low-Code Development）的一种尝试，并根据后期应用场景迭代优化，可以初步实现“业务自提需求，自构建”的目标；业务人员和技术人员可以基于同一技术视角沟需求，减少对需求的理解偏差

■配置工具生成文件为：

config_public.h: 为静态库 export 声明头文件，包含如下内容（代码片段根据图 5 呈现拓扑结构生成）

①对矩阵容量的声明

```
#define SUBNODES_PER_POOL  3
#define CONTACTORS_PER_NODE  3
#define MAX_PLUGIN_NUM  9
```

代码 1

②功率池 ID 序号枚举约定

```
typedef enum{
PWRPOOL_UNKNOWN = 0,
PWRPOOL1,
PWRPOOL2,
```

```
PWRPOOL3,
PWRPOOL_MAX=PWRPOOL3
}PWRPOOL_ID;
```

📜代码 2

③功率节点 ID 序号枚举约定，

功率节点 ID 序号=（功率池序号-1）✕CONTACTORS_PER_NODE+

```
 typedef enum{
PWRNODE_UNKNOWN = 0,
PWRNODE1_1,
PWRNODE1_2,
PWRNODE1_3,
PWRNODE2_1,
PWRNODE2_2,
PWRNODE2_3,
PWRNODE3_1,
PWRNODE3_2,
PWRNODE3_3,
PWRNODE_MAX=PWRNODE3_3
}PWRNODE_ID;
```

📜代码 3

④接触器节 ID 点序号枚举约定，由于配置工具默认遵循**每个功率池均配置实际应用中容量最大池所包含的功率节点数目**，并且每个接触器只从属于一个功率模块，所以可通过接触器 ID 序号反向获取节点 ID 序号和功率池 ID 序号

```
 typedef enum{
CONTACTOR_UNKNOWN = 0,
CONTACTOR1_1_1,
CONTACTOR1_1_2,
CONTACTOR1_1_3,
CONTACTOR1_2_1,
CONTACTOR1_2_2,
CONTACTOR1_2_3,
CONTACTOR1_3_1,

...

CONTACTOR3_2_3,
CONTACTOR3_3_1,
CONTACTOR3_3_2,
CONTACTOR3_3_3,
CONTACTOR_MAX=CONTACTOR3_3_3
}CONTACTOR_ID;
```

📜代码 4

⛔IMSU-X 调用程序在接口函数返回值和形参上必须与该头文件保持一致

👉**config_private.h** 为静态库自用声明头文件，包含如下内容

① 与每个功率模块配备接触器数目相关的结构体和过程执行逻辑的自举宏

```
#ifdef PICKOUT_CONTACTANT
#pragma message("structure and initializing deeds abt contactor introduced here...")
CONTACTEE(1)
CONTACTEE(2)
CONTACTEE(3)
#undef PICKOUT_CONTACTANT
#endif
```

📜代码 5

② 描述功率节点间、枪头与功率节点间电气联系的自举宏

```
#ifdef PICKOUT_MUTUALREF
```

```
#pragma message("compiling nodes heuristic linkage ...")
CONN_NODE(1, 1, 1)
CONN_NODE(1, 1, 2, 1)
CONN_NODE(1, 1, 3, 2)
CONN_NODE(2, 2, 1)
CONN_NODE(2, 2, 2, 1)
CONN_NODE(2, 2, 3, 2)
CONN_NODE(3, 3, 1)
CONN_NODE(3, 3, 2, 1)
CONN_NODE(3, 3, 3, 2)
CONN_NODE(4, 1, 4)
CONN_NODE(4, 1, 5, 4)
CONN_NODE(4, 1, 6, 5)
CONN_NODE(5, 2, 4)
CONN_NODE(5, 2, 5, 4)
CONN_NODE(5, 2, 6, 5)
CONN_NODE(6, 3, 4)
CONN_NODE(6, 3, 5, 4)
CONN_NODE(6, 3, 6, 5)
CONN_NODE(7, 1, 7)
CONN_NODE(7, 1, 8, 7)
CONN_NODE(7, 1, 9, 8)
CONN_NODE(8, 2, 7)
CONN_NODE(8, 2, 8, 7)
CONN_NODE(8, 2, 9, 8)
CONN_NODE(9, 3, 7)
CONN_NODE(9, 3, 8, 7)
CONN_NODE(9, 3, 9, 8)
#undef PICKOUT_MUTUALREF
#endif
```

代码 6

分配策略

■静态库程序在流程上大致为“配节点”→“找节点”→“选节点”三个步骤，选取节点即功率分配决策的最终实现，功率分配算法模块是整个静态库的**智能核心**，负责根据实时充电需求、系统可用功率资源和预设策略，计算最优的功率分配方案。算法设计采用**自适应加权轮询调度**与**优先级预分配**相结合的策略，确保系统既能公平服务所有连接车辆，又能优先满足重要用户或紧急需求。

算法输入包括：

- 各充电枪实时功率需求
- 各电源模块可用功率容量
- 系统当前功率负载情况
- 预设分配策略参数

算法输出包括：

- 指定功率节点输出功率
- 继电器矩阵切换控制序号及动作序列
- 功率分配时间调度表（可选）

算法执行流程包括以下步骤：

1. **需求收集**：获取各充电枪的功率需求，区分基本需求与可变需求，同时标注枪头为液冷超充枪还是普通枪
2. **资源评估**：计算各电源模块的可用功率容量，考虑温度降额和效率因素
3. **策略匹配**：根据系统运行模式选择适当的分配策略（均等分配、优先级分配、最大需求优先等）

分配策略如以下定义：

```
typedef enum
{
    CRITERION_NONE = 0,
    CRITERION_PRIOR,          // 车优先级
```

```
    CRITERION_MIN_COST,       // 经济运行
    CRITERION_MAX_POWER,      // 最大需求
    CRITERION_EQU_SANITY,     // 均衡健康度
    CRITERION_LMT_CAPACITY,   // 单铜排限电流容量
    CRITERION_MAX_EFFICIENCY, // 模块效率
} CRITERION;
```

代码 7

4. **方案生成**：计算功率分配方案，确保系统总功率不超限且分配效率最优，限定条件包括总功率、功率池功率限定、电气汇流点或接触器铜排电流限制等
5. **安全校验**：检查方案是否符合安全约束条件，防止过载和过热
6. **指令生成**：输出继电器 ID 序号和功率值给调用程序，用来生成具体的控制指令序列，包括继电器切换、预充电和功率模块调节指令

算法考虑**多枪充电场景**下的动态调整能力，当有新充电枪接入或现有充电枪需求变化时，系统能够在 100ms 内重新计算分配方案，实现平滑的功率转移

接口定义

功率节点控制输出接口函数

```
<Import>被静态库调用，由 IMSU-X 提供
@parameter1：接触器节点 ID，可以由此确定功率节点 ID
@parameter2：功率节点预设电压
@parameter3：功率节点预设电流
@return：功率节点预设动作结果返回码

ReturnCode set_Pwrnode_Output(uint8_t contactor_id, float voltage, float current);
```

代码 8

功率节点信息获取接口函数

```
typedef enum
{
    UNINITIALIZED = 0,
    READY,
    FAULT,
    OCUPPIED,
    DISCARDED,
    UNKNOWN
} NODE_STATE;
<Import>被静态库调用，由 IMSU-X 提供
@parameter：接触器节点 ID，可以由此确定功率节点 ID
@return：功率节点当前状态

NODE_STATE get_Pwrnode_Info(uint8_t contactor_id);
```

代码 9

任务入口函数

```
<export>被 IMSU-X 调用；静态库入口会占用系统一个任务资源来处理分配业务
@parameter：void* pdealing 含义待定
@return：void
```

```
    void Task_PwrAlloc(void* pdealing);
```

代码 10

任务更新回调函数

```
    <export>被 IMSU-X 调用；静态库任务周期运行会回调此函数，用来延时让出资源及投喂任务内软狗
    @parameter：void
    @return：void

    NodeStatus recall_Task_update(void);
```

代码 11

功率分配功能模块状态获取

```
    <export>被 IMSU-X 调用；静态库告知模块是否可用，静态库采用安全设计，如发现静态库专有内存区出现数据段被破坏、栈溢出等异常，会不再继续执行流程，任务仅反复空跑并告知 IMSU-X 调用程序当前功率分配功能模块不可用
    @parameter：void
    @return：当前功率分配功能模块是否可用

    bool is_PwrAlloc_Available(void);
```

代码 12

extern 变量索引

```
    <import>IMSU-X 程序将以下外引变量通过链接器符号表告知静态库
    可以提供功率节点信息的变量，包括但不限于以下伪代码提供的信息
    typedef enum
{
    MOD_INIT = 0,   // 模块初始化
    MOD_STANDBY,    // 模块待机
    MOD_CHARGING,   // 模块充电中
    MOD_FAULT,      // 模块故障
    MOD_MAINTENANCE // 模块维护
} ModuleStatus;
typedef struct
{
    uint8_t pwrnode_id;   // 模块标识
    float max_voltage;   // 最大输出电压
    float max_current;   // 最大输出电流
    float max_power;     // 最大输出功率
    float current_power; // 当前输出功率
    float temperature;   // 模块温度
    ModuleStatus status; // 模块状态
} PowerSupply;

    可以提供枪头信息的变量，包括但不限于以下伪代码提供的信息
typedef enum
{
    PLUGIN_UNPLUGGED = 0, // 未插入
    PLUGIN_PLUGGED,       // 已插入未充电
    PLUGIN_CHARGING,      // 充电中
    PLUGIN_SUSPENDED,     // 充电暂停
    PLUGIN_FAULT          // 故障
} PLuginStatus;

typedef struct
{
    uint8_t plugin_id;   // 充电枪标识
    float voltage_req;   // 电压需求
    float current_req;   // 电流需求
    float power_req;     // 功率需求
```

```
    uint8_t priority;    // 优先级
    PluginStatus status; // 充电枪状态
} PowerDemand;
```

代码 13

代码安全

■由于电气拓扑结构运行中可变，软件中对应的数据采用 C99 弹性数组实现

> 16 As a special case, the last element of a structure with more than one named member may have an incomplete array type; this is called a *flexible array member*. With two exceptions, the flexible array member is ignored. First, the size of the structure shall be equal to the offset of the last element of an otherwise identical structure that replaces the flexible array member with an array of unspecified length.[106] Second, when a . (or ->) operator has a left operand that is (a pointer to) a structure with a flexible array member and the right operand names that member, it behaves as if that member were replaced with the longest array (with the same element type) that would not make the structure larger than the object being accessed; the offset of the array shall remain that of the flexible array member, even if this would differ from that of the replacement array. If this array would have no elements, it behaves as if it had one element but the behavior is undefined if any attempt is made to access that element or to generate a pointer one past it.

援引 ISO/IEC9899:1999

■在 IAR 编译环境下设置一个专有的 RAM 区域给功率分配功能模块，与 IMSU-X 调用程序使用的 RAM 区域隔离，在编译静态链接库的布局文件.icf 中划定一块 RAM 区域

```
define symbol __ICFEDIT_region_CUSTOM_RAM_start__  = 0x20000000;
define symbol __ICFEDIT_region_CUSTOM_RAM_length__ = 0x00001000; // 4kB
define symbol __ICFEDIT_region_CUSTOM_HEAP_length__ = 0x00002000; // 8kB

define symbol __PDU_CORE_RAM_start__ = __ICFEDIT_region_CUSTOM_RAM_start__;
define symbol __PDU_CORE_RAM_end__   = __ICFEDIT_region_CUSTOM_RAM_start__ + __ICFEDIT_region_CUSTOM_RAM_length__;

...
define symbol __PDU_CORE_HEAP_start__ = __PDU_CORE_RAM_end__;
define symbol __PDU_CORE_HEAP_end__   = __PDU_CORE_HEAP_start__ + __ICFEDIT_region_CUSTOM_HEAP_length__;
define symbol __PDU_CORE_HEAP_size__  = __ICFEDIT_region_CUSTOM_HEAP_length__;


define region PDU_CORE_RAM_region = mem:[from __PDU_CORE_RAM_start__ to __PDU_CORE_RAM_end__];
place in PDU_CORE_RAM_region { section .pdu_core_section };
...

export symbol __PDU_CORE_HEAP_start__;
export symbol __PDU_CORE_HEAP_end__;
export symbol __PDU_CORE_HEAP_size__;
```

代码 14

```
#pragma location = ". pdu_core_section "
const volatile uint32_t  dummy_value ;

#pragma location = ". pdu_core_section"
void critical_function(void) {
    // Heat maps high coverage code
}
...
extern const uint8_t __PDU_CORE_HEAP_start__[];
extern const uint8_t __PDU_CORE_HEAP_end__[];
extern const uint32_t __PDU_CORE_HEAP_size__;
```

```
void* get_custom_pool_start(void) {
    return (void*)__PDU_CORE_HEAP_start__;
}

size_t get_PDU_CORE_HEAP_size(void) {
    return (size_t)__PDU_CORE_HEAP_size__;
}

/* plain malloc */

void* custom_malloc(size_t size) {

static uint8_t* PDU_CORE_HEAP_ptr = (uint8_t*)__PDU_CORE_HEAP_start__;
uint8_t* current = PDU_CORE_HEAP_ptr;
size_t used_size = current - __PDU_CORE_HEAP_start__;
size_t remaining = __PDU_CORE_HEAP_size__ - used_size;

    if (size > remaining) {
        return NULL;
    }

    PDU_CORE_HEAP_ptr += size;
    return current;
}
```

代码 15

■为防止专有 RAM 内数据被其他内存区数据破坏（本方案中如果专有内存区置于 0x20000000，则主要防止上溢），在几个关键数据结构体中内置保护值

```
#define FRONT_MAGICWORD 0xDEADCAFE
#define REAR_MAGICWORD 0xBABEFACE
#define GET_REAR_CANARY_PTR(ptr, type) \
    ((unsigned int *)((char *)(ptr) + sizeof(type) + (ptr)->length * sizeof(((type *)0)->obj_array[0])))
#define IS_FRONT_CANARY_INTACT(ptr, type) \
    ((ptr) ? (((const type *)(ptr))->front_canary == (FRONT_MAGICWORD)) : false)
#define GET_REAR_CANARY_PTR(ptr, type) \
    ((unsigned int *)((char *)(ptr) + sizeof(type) + (ptr)->length * sizeof(((type *)0)->obj_array[0])))
#define IS_REAR_CANARY_INTACT(ptr, type) \
    ((ptr) ? (*GET_REAR_CANARY_PTR(ptr, type) == (REAR_MAGICWORD)) : false)

typedef struct
{
    unsigned int front_canary; // absolutely required to be top element
    size_t length;
    Alloc_nodeObj obj_array[];
} Alloc_nodeArray;

typedef struct
{
    unsigned int front_canary; // absolutely required to be top element
    size_t length;
    Alloc_pluginObj obj_array[];
} Alloc_pluginArray;

...

bool hear_Canaries_Twittering(void)
{
    if (!IS_FRONT_CANARY_INTACT(gpNodesArray, Alloc_nodeArray))
    {
        printf("gpNodesArray Front canary corrupted!\r\n");
        return false;
    }

    if (!IS_REAR_CANARY_INTACT(gpNodesArray, Alloc_nodeArray))
    {
        printf("gpNodesArray Rear canary corrupted!\r\n");
        return false;
    }

    if (!IS_FRONT_CANARY_INTACT(gpPluginsArray, Alloc_pluginArray))
    {

        printf("gpPluginsArray Front canary corrupted!\r\n");
        return false;
```

```
    }
    if (!IS_REAR_CANARY_INTACT(gpPluginsArray, Alloc_pluginArray))
    {
        printf("gpPluginsArray Rear canary corrupted!\r\n");
        return false;
    }
    return true;
```

代码 16

■软件调试稳定后酌情添加栈级保护，提高代码健壮性

--stack_protection

Syntax	`--stack_protection`
Description	Use this option to enable stack protection for the functions that are considered to need it.
See also	*Stack protection*, page 92.

Project>Options>C/C++ Compiler>Code>Stack protection

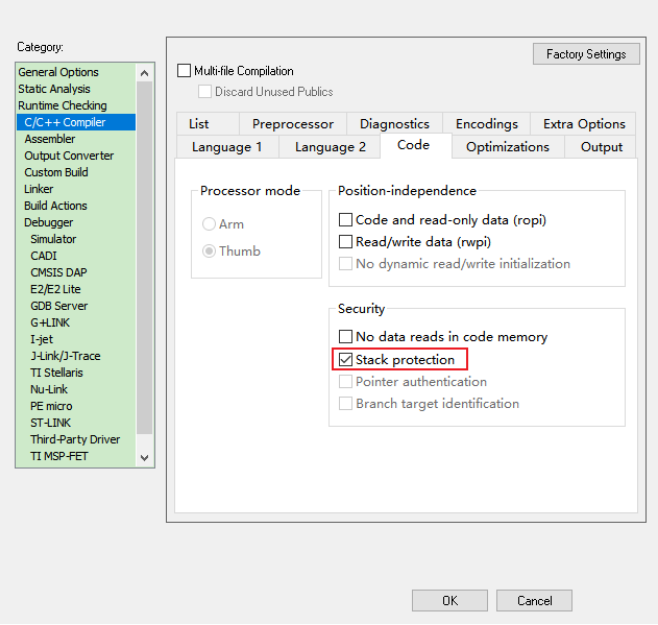

图 7

```
/*    your\IAR\path\arm\src\lib\runtime\stack_protection.c  */
#include <stdlib.h>      /* For void abort(void) */
#include <stdint.h>      /* For uint32_t */

#pragma language=extended

void __init_stack_chk_guard(void);

/*
 * This variable holds the value that the canary will be initialized with
 * and also compared with before returning from the function.
 */
#pragma required=__init_stack_chk_guard
__no_init uint32_t __stack_chk_guard;

/*
 * The __stack_chk_fail(void) function is called when a modified canary is detected.
 */
#pragma no_stack_protect
__attribute__((noreturn)) void __stack_chk_fail(void)
{
  /*
```

```
  LogStackSmashed();
  */
  abort();
}

/*
 * This function gets called at startup as part of construction of static C++ objects
 * and is a way to programatically initialize the __stack_chk_guard when the system starts.
 */
#pragma early_dynamic_initialization
#pragma no_stack_protect
__attribute__((constructor)) void __init_stack_chk_guard(void)
{
  __stack_chk_guard = 4711;
}
```

代码 17

工况可视

■静态库提供运行工况输出，关联 RTT 第 2 通道，调试过程中可通过直接调用或设置持续输出模式全息观察功率分配逻辑和分配结果

📌以下图所示（4*功率池-6*节点-5*连接器）的拓扑结构为例

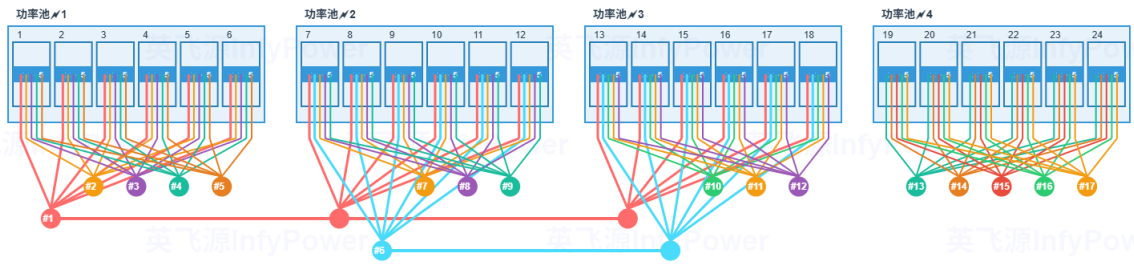

图 8

静态库会生成相应拓扑示意图，并实时直观显示功率节点运行数据、接触器投切状态、枪头功率需求等，并将功率分配策略轨迹实时输出 log 供调试分析

👉静态库同时提供 tracelog 打印日志，可还原事件因果链，帮助确认是否按预期运行；例如某次功率分配失败是否由资源评估错误或返回失败导致

👉静态库兼顾 IMSU-X 系统资源（如存储空间、实时性），提供合理日志粒度、存储方式的 tracelog 接口

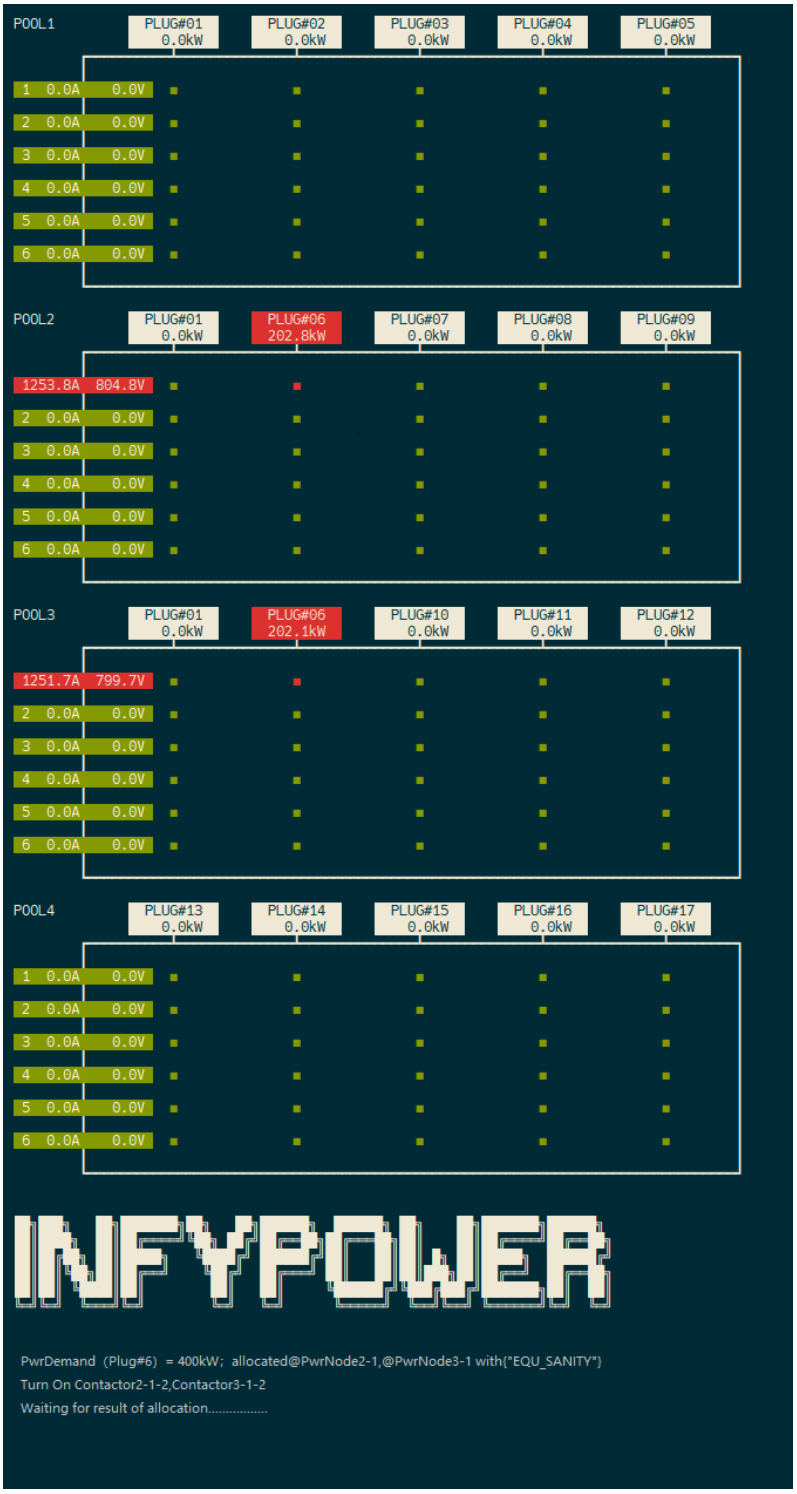
POOL1
PLUG#01 0.0kW
PLUG#02 0.0kW
PLUG#03 0.0kW
PLUG#04 0.0kW
PLUG#05 0.0kW
1 0.0A 0.0V
2 0.0A 0.0V
3 0.0A 0.0V
4 0.0A 0.0V
5 0.0A 0.0V
6 0.0A 0.0V
POOL2
PLUG#01 0.0kW
PLUG#06 202.8kW
PLUG#07 0.0kW
PLUG#08 0.0kW
PLUG#09 0.0kW
1253.8A 804.8V
2 0.0A 0.0V
3 0.0A 0.0V
4 0.0A 0.0V
5 0.0A 0.0V
6 0.0A 0.0V
POOL3
PLUG#01 0.0kW
PLUG#06 202.1kW
PLUG#10 0.0kW
PLUG#11 0.0kW
PLUG#12 0.0kW
1251.7A 799.7V
2 0.0A 0.0V
3 0.0A 0.0V
4 0.0A 0.0V
5 0.0A 0.0V
6 0.0A 0.0V
POOL4
PLUG#13 0.0kW
PLUG#14 0.0kW
PLUG#15 0.0kW
PLUG#16 0.0kW
PLUG#17 0.0kW
1 0.0A 0.0V
2 0.0A 0.0V
3 0.0A 0.0V
4 0.0A 0.0V
5 0.0A 0.0V
6 0.0A 0.0V
INFYPOWER
PwrDemand (Plug#6) = 400kW; allocated@PwrNode2-1,@PwrNode3-1 with{"EQU_SANITY"}
Turn On Contactor2-1-2,Contactor3-1-2
Waiting for result of allocation.....

图 9