ADC\_DIAG\_SWC组件软件详细设计







|  |  |  |  |
| --- | --- | --- | --- |
| 作者 | 肖沣益 | 日期 | 2024.01.10 |
| 审核 |  | 日期 |  |
| 批准 |  | 日期 |  |

**苏州天准科技股份有限公司**

**文件变更履历**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **修改日期** | **版本** | **修改内容** | **作者** | **审核** | **批准** |
| 2024/01/10 | V1.0 | 1、初建文档。 | 肖沣益 |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**目录**

[1. 概要 7](#_Toc1338987534)

[1.1. 目的 7](#_Toc1609642757)

[1.2. 范围 7](#_Toc2054621861)

[1.3. 参考文件 7](#_Toc1299905665)

[1.4. 用语、缩略语等定义 7](#_Toc481550740)

[2. 假设和约束 7](#_Toc287982849)

[2.1. 描述软件组件的假设和约束 7](#_Toc620061783)

[3. 软件组件设计 11](#_Toc197466096)

[3.1. 组件静态设计 11](#_Toc1221207071)

[3.2. 软件组件ASIL分配 16](#_Toc294627035)

[3.3. 组件动态设计 17](#_Toc552646290)

[4. 资源消耗目标的实现策略 18](#_Toc2101537347)

[4.1. 为满足RAM资源消耗要求 18](#_Toc742404563)

[4.2. 为满足flash资源消耗要求 18](#_Toc641790129)

[5. 组件接口 18](#_Toc1129270633)

[5.1外部接口 18](#_Toc1854740083)

[5.2内部接口 19](#_Toc34623140)

[6. 一般定义 20](#_Toc1672490258)

[6.1. 标识符预处理 20](#_Toc319193297)

[6.2. 类型重定义 21](#_Toc1303683685)

[6.3. 全局变量 21](#_Toc2100427402)

[6.4. 静态变量 22](#_Toc1662277810)

[7. 功能单元详细信息 24](#_Toc1004071123)

[7.1. 功能函数: static void diag\_lock\_init(void) 24](#_Toc741650773)

[7.2. 功能函数:static void diag\_print\_ratelimit\_init(void) 24](#_Toc1946753059)

[7.3. 功能函数:static int32\_t diag\_polling\_check\_service\_init(void) 25](#_Toc535851399)

[7.4. 功能函数:static int32\_t diag\_resource\_init(void) 26](#_Toc116676926)

[7.5. 功能函数:static void diag\_resource\_deinit(void) 27](#_Toc1522762654)

[7.6. 功能函数:static void diag\_chrdev\_deinit(struct diag \*diag) 28](#_Toc404860150)

[7.7. 功能函数:static void diag\_netlink\_deinit(struct diag \*diag) 29](#_Toc926614786)

[7.8. 功能函数:const uint8\_t \* fm\_name(uint16\_t module\_id) 30](#_Toc124561794)

[7.9. 功能函数:static int32\_t diag\_register\_arg\_check(const struct diag\_register\_info \*register\_info) 31](#_Toc1743847685)

[7.10. 功能函数:static int32\_t diag\_check\_module\_register(const struct diag\_register\_info \*register\_info, struct diag\_module \*module) 32](#_Toc388773895)

[7.11. 功能函数:static void diag\_register\_record(uint16\_t module\_id) 33](#_Toc31700007)

[7.12. 功能函数:static int32\_t diag\_check\_send\_condition(struct diag\_event\_id \*event\_id, struct diag\_event \*event) 34](#_Toc896269702)

[7.13. 功能函数:static int32\_t diag\_send\_arg\_check(struct diag\_event \*event, int8\_t \*register\_state) 35](#_Toc870324635)

[7.14. 功能函数:int32\_t diag\_polling\_check\_service\_switch(int32\_t switch\_flag) 36](#_Toc319682856)

[7.15. 功能函数:static void diag\_polling\_check\_service\_deinit(void) 37](#_Toc1516331486)

[7.16. 功能函数:static int diag\_register\_check\_record(uint16\_t module\_id) 38](#_Toc1067790731)

[7.17. 功能函数:struct diag\_module \*diag\_module\_event\_register\_or\_not(struct diag\_event \*event, int32\_t \*event\_idx, uint8\_t status) 39](#_Toc1540889927)

[7.18. 功能函数:static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag) 40](#_Toc1810958521)

[7.19. 功能函数:static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag) 41](#_Toc1620437022)

[7.20. 功能函数:static int32\_t diag\_register\_module\_to\_list(const struct diag\_register\_info \*register\_info) 42](#_Toc1494943626)

[7.21. 功能函数:static int32\_t diag\_insert\_to\_priority\_list(struct diag\_module\_event \*module\_event, struct diag\_event \*event) 43](#_Toc405879436)

[7.22. 功能函数:static int32\_t diag\_module\_event\_add\_queue(struct diag\_event \*event) 44](#_Toc114743503)

[7.23. 功能函数:static void handle\_and\_move(const struct list\_head \*prio\_list, raw\_spinlock\_t \*lock) 46](#_Toc476730611)

[7.24. 功能函数:static int32\_t diag\_handle\_cb(struct diag\_event \*event, uint32\_t errcode, uint32\_t critical) 47](#_Toc113135871)

[7.25. 功能函数:static void diag\_work\_handler(struct work\_struct \*work) 48](#_Toc149366643)

[7.26. 功能函数:static void diag\_execute\_polling\_event(void) 49](#_Toc1737222)

[7.27. 功能函数:static int32\_t diag\_polling\_case\_thread(void \*arg) 50](#_Toc432329168)

[7.28. 功能函数:int32\_t diag\_is\_ready(void) 51](#_Toc1453050329)

[7.29. 功能函数:int8\_t diagnose\_get\_register\_state(uint16\_t module\_id) 52](#_Toc2102164624)

[7.30. 功能函数:int32\_t diagnose\_register(const struct diag\_register\_info \*register\_info) 54](#_Toc2094606978)

[7.31. 功能函数:int32\_t diagnose\_send\_event(struct diag\_event \*event) 55](#_Toc309637804)

[7.32. 功能函数:void diagnose\_get\_idle\_node\_number(uint16\_t \*number) 56](#_Toc696331749)

[7.33. 功能函数:static int diag\_probe(struct platform\_device \*pdev) 57](#_Toc1893876390)

[7.34. 功能函数:static int diag\_remove(struct platform\_device \*pdev) 58](#_Toc845489204)

[8. 其他 60](#_Toc813008675)

# 概要

## 目的

本文件的目的是描述ADC\_DIAG\_SWC组件的软件详细设计；软件详细设计是软件单元构建的输入。

## 范围

本组件软件详细设计主要定义限制与约束，软件组件设计（静态、动态设计），以及资源消耗目标的实现策略，组件接口以及一般定义等。

本计划中记录的活动、工具、规则和概念适用于CD82项目。

## 参考文件

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **编号** | **参考文档** | **版本** | **路径** | **发行者** |
| 1 | X9\_Processor\_TRM\_Rev01.01 | V1.01 | [芯驰客户支持](https://support.semidrive.com/) | 芯驰 |
| 2 | X9U\_Processor\_Datasheet\_Rev00.03 | V0.03 | [芯驰客户支持](https://support.semidrive.com/) | 芯驰 |
| 3 |  |  |  |  |
| 4 |  |  |  |  |

## 用语、缩略语等定义

|  |  |  |
| --- | --- | --- |
| **编号** | **用语、缩写语** | **含义、定义和正式名称** |
| 1 | 组件component | 软件“组件”是软件架构的最低层级的“要素”，以定义最终的详细设计。一个软件“组件”可包含一个或多个软件“单元” |
| 2 | 单元 unit | 不能被进一步分解的软件组件 |
| 3 | DIIAG | 诊断模块 |

# 假设和约束

## 描述软件组件的假设和约束

举例：

1. 资源消耗的目标要求

|  |  |
| --- | --- |
| 资源 | 要求 |
| Flash | 大小不超过100k |
| Ram | 大小不超过1M |
| CPU负载率 | 不超过1% |

1. 编码的约束：

|  |  |
| --- | --- |
| 编号 | 约束 |
| 1 | 编号规范符合MISRA C: 2023的规则 |
| 2 | 函数命名规则 |
|  |  |
|  |  |

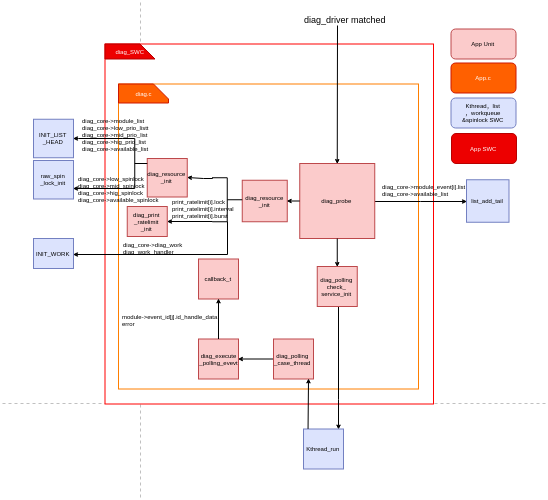
3）单元设计的准则等

# 软件组件设计

从不同视角详细描述软件元素

## 组件静态设计

描述软件组件的技术架构（框图和部署图）。



软件组件静态架构图

| **编号** | **.C文件名称** | **概要** | **备注** |
| --- | --- | --- | --- |
| Comp-FU-001 | diag diag\_fs diag\_netlink diag\_communication | X9U诊断相关功能实现 |  |
| Comp-FU-002 | sdrv\_wdg\_stl | 对WDT 模块进行功能安全检测 |  |
| Comp-FU-003 | sfmu | 支持统一上报接口SDK |  |
| Comp-FU-004 | ttl | 支持A55 STL驱动检测 |  |
| Comp-FU-005 | ttl | 支持A55 STL故障注入 |  |
| Comp-FU-006 | mipicsi-stl | 对CSIMIPI 模块进行功能安全检测 |  |
| Comp-FU-007 | tztek\_phy\_stl | 对PHY模块进行功能安全检测 |  |
| Comp-FU-008 | gpio-stl | 对GPIO模块进行功能安全检测 |  |

| **编号** | **功能单元** | **.C名称** | **概要** | **备注** |
| --- | --- | --- | --- | --- |
| Comp-FU-001-01 | diag\_lock\_init | diag | 初始化diag\_core结构体的链表成员，及自旋锁成员 |  |
| Comp-FU-001-02 | diag\_print\_ratelimit\_init | diag | 初始化存储和管理限流状态的相关信息 |  |
| Comp-FU-001-03 | diag\_polling\_check\_service\_init | diag | 创建一个新的线程来执行  26函数 |  |
| Comp-FU-001-04 | diag\_resource\_init | diag | 初始化诊断资源，调用1，2，将24加到diag\_core工作队列成员中 |  |
| Comp-FU-001-05 | diag\_resource\_deinit | diag | 释放诊断资源 |  |
| Comp-FU-001-06 | diag\_chrdev\_deinit | diag | 程序退出时清理字符设备的资源 |  |
| Comp-FU-001-07 | diag\_netlink\_deinit | diag | 释放网络链接相关的资源，包括关闭套接字描述符和销毁互斥锁 |  |
| Comp-FU-001-08 | Fm\_name | diag | 根据给定的ID返回响应的模块名称 |  |
| Comp-FU-001-09 | diag\_register\_arg\_check | diag | 检测diag\_register\_info成员是否满足特定的条件 |  |
| Comp-FU-001-10 | diag\_check\_module\_register | diag | 检查诊断模块的注册情况，包括模块ID和事件ID的重复注册检查，以及将新注册的事件信息添加到已注册模块中 |  |
| Comp-FU-001-11 | diag\_register\_record | diag | 注册诊断记录的模块，但仅在模块ID不重复时才执行注册操作，并使用自旋锁确保线程安全 |  |
| Comp-FU-001-12 | diag\_check\_send\_condition | diag | 检测是否满足条件，满足设置事件最后状态和最后发送时间 |  |
| Comp-FU-001-13 | diag\_send\_arg\_check | diag | 检查和处理传入的事件结构体是否合法，如果合法则根据一定条件进行进一步处理调用12，17 |  |
| Comp-FU-001-14 | diag\_polling\_check\_service\_switch | diag | 检查和控制轮训诊断服务的开关状态 |  |
| Comp-FU-001-15 | diag\_polling\_check\_service\_deinit | diag | 控制轮训诊断服务的开关关闭 |  |
| Comp-FU-001-16 | diag\_register\_check\_record | diag | 检查给定module\_id是否已经在记录中注册，如果注册了则返回0，如果未注册则返回-1 |  |
| Comp-FU-001-17 | diag\_module\_event\_register\_or\_not | diag | 主要作用是在诊断模块列表中查找特定事件，如果找到则注册该事件，并返回对应的模块指针和事件索引。如果未找到匹配的模块或事件，则返回NULL |  |
| Comp-FU-001-18 | diag\_show\_event\_info | diag | 输出的调试信息 |  |
| Comp-FU-001-19 | diag\_register\_module\_to\_list | diag | 创建一个新的诊断模块，并将其相关信息添加到模块列表中 |  |
| Comp-FU-001-20 | diag\_insert\_to\_priority\_list | diag | 事件结构体 event 插入到不同优先级队列中，并根据事件的优先级进行分类和处理 |  |
| Comp-FU-001-21 | diag\_module\_event\_add\_queue | diag | 事件结构体 event 插入到不同优先级队列中，调用20 |  |
| Comp-FU-001-22 | diag\_handle\_cb | diag | 通过传入的诊断事件，去调用与之匹配的回调函数 |  |
| Comp-FU-001-23 | handle\_and\_move | diag | 遍历一个链表，处理每个元素的信息，并将处理完的元素移动到另一个链表module\_event |  |
| Comp-FU-001-24 | diag\_work\_handler | diag | 调用23，给模块事件分优先级 |  |
| Comp-FU-001-25 | diag\_execute\_polling\_event | diag | 遍历模块列表中的每个模块，然后遍历每个模块的事件列表，如果事件的类型是EventIdPoll且有回调函数，就执行回调函数并传递相应参数 |  |
| Comp-FU-001-26 | diag\_polling\_case\_thread | diag | 创建线程，3s循环执行25 |  |
| Comp-FU-001-27 | diag\_is\_ready | diag | 检查某个诊断驱动程序是否已经初始化完成 |  |
| Comp-FU-001-28 | diagnose\_register | diag | 1. 调用09检查参数有效 2. 调用10检查模块是否注册，没有就注册 3. 调用19将模块添加链表中 4. 调用16记录注册的模块 |  |
| Comp-FU-001-29 | diagnose\_unregister | diag | 则注销指定的模块 |  |
| Comp-FU-001-30 | diagnose\_send\_event | diag | 1. 调用16检查事件对应模块是否注册 2. 调用13对事件参数检查 3. 调用22去执行回调函数 4. 调用18显示事件信息 5. 调用21将事件添加队列   6.调schedule\_work 函数安排一个工作项，以便稍后异步处理事件 |  |
| Comp-FU-001-31 | diagnose\_get\_idle\_node\_number | diag | 检查诊断驱动程序的初始化状态和链表中的节点数目 |  |
| Comp-FU-001-32 | diag\_probe | diag | 诊断匹配成功后加载函数 |  |
| Comp-FU-001-33 | diag\_remove | diag | 诊断驱动移除 |  |
| Comp-FU-001-34 | diagnose\_get\_register\_state | diag | 检测诊断登记函数 |  |
| Comp-FU-002-01 | wdtstl\_show | sdrv\_wdg\_stl | cat查看故障注入文件系统节点时打印help信息 |  |
| Comp-FU-002-02 | wdtstl\_store | sdrv\_wdg\_stl | 向故障注入文件系统节点echo值的时候的处理接口 |  |
| Comp-FU-002-03 | wdgstl\_sysfs\_node\_create | sdrv\_wdg\_stl | 创建watchdog 故障注入文件系统节点 |  |
| Comp-FU-002-04 | notify\_wdg\_stl | sdrv\_wdg\_stl | 模块内部进行实践信息发送的接口 |  |
| Comp-FU-002-05 | wdg\_stl\_polling\_callback | sdrv\_wdg\_stl | watchdog周期性诊断入口函数 |  |
| Comp-FU-002-06 | wdg\_inject\_callback | sdrv\_wdg\_stl | 面向diag的watchdog故障注入接口 |  |
| Comp-FU-002-07 | deal\_work\_handler | sdrv\_wdg\_stl | 异步事件处理队列的handler |  |
| Comp-FU-002-08 | register\_work\_handler | sdrv\_wdg\_stl | 异步工作队列，向diag注册模块的handler |  |
| Comp-FU-002-09 | resource\_init | sdrv\_wdg\_stl | 资源初始化：初始化自旋锁、申请内存、初始化链表、绑定私有数据、初始化工作队列 |  |
| Comp-FU-002-10 | register\_init | sdrv\_wdg\_stl | 填充向diag申请注册时使用的数据 |  |
| Comp-FU-002-11 | watchdog\_register\_read\_back\_stl | sdrv\_wdg\_stl | read\_back故障注入 |  |
| Comp-FU-002-12 | wdg\_stl\_event\_debug | sdrv\_wdg\_stl | debug测试：打印事件信息 |  |
| Comp-FU-002-13 | wdg\_stl\_register\_callback | sdrv\_wdg\_stl | 冗余：绑定wdt模块与wdt->cb，用于wdt发送事件 |  |
| Comp-FU-002-14 | wdg\_stl\_start | sdrv\_wdg\_stl | ops成员入口：向diag注册模块 |  |
| Comp-FU-002-15 | wdg\_stl\_stop | sdrv\_wdg\_stl | ops 操作函数成员：关闭事件上报队列 |  |
| Comp-FU-002-16 | wdg\_stl\_check\_reg | sdrv\_wdg\_stl | wdt回读检查 |  |
| Comp-FU-002-17 | wdg\_stl\_init | sdrv\_wdg\_stl | wdt-stl初始化入口，供wdt调用。申请内存、绑定参数、初始化flag |  |
| Comp-FU-002-18 | wdg\_stl\_deinit | sdrv\_wdg\_stl | 释放stl相关内存 |  |
| Comp-FU-006-01 | mipicsi\_stl\_check\_reg | mipicsi-stl | 检查关键寄存器的值是否与标准值保持一致 |  |
| Comp-FU-006-02 | mipicsi\_stl\_notify | mipicsi-stl | 填充诊断上报事件，发起异步上报 |  |
| Comp-FU-006-03 | create\_fs | mipicsi-stl | 创建故障注入文件系统节点 |  |
| Comp-FU-006-04 | mipicsi\_stl\_rback\_inject | mipicsi-stl | 故障注入接口：read\_back |  |
| Comp-FU-006-05 | mipicsi\_stl\_interval\_rback\_inject | mipicsi-stl | 故障注入接口：period reg check |  |
| Comp-FU-006-06 | mipicsi\_stl\_add\_to\_list | mipicsi-stl | 将事件添加到待发送链表 |  |
| Comp-FU-006-07 | mipicsi\_stl\_send\_diag | mipicsi-stl | 发送待发送链表的第一个时间，发送后释放内存 |  |
| Comp-FU-006-08 | mipicsi\_stl\_workqueue | mipicsi-stl | 异步工作队列：发送事件 |  |
| Comp-FU-006-09 | register\_work\_handler | mipicsi-stl | 异步延时工作队列：向diag注册module |  |
| Comp-FU-006-10 | mipicsi\_stl\_polling\_callback | mipicsi-stl | diag周期性诊断服务的回调函数 |  |
| Comp-FU-006-11 | mipicsi\_stl\_RB\_inject\_callback | mipicsi-stl | diag故障注入回调：read\_back |  |
| Comp-FU-006-12 | mipicsi\_stl\_PRB\_inject\_callback | mipicsi-stl | diag故障注入回调：period reg check |  |
| Comp-FU-006-13 | register\_init | mipicsi-stl | 填充向diag申请注册的结构体内容 |  |
| Comp-FU-006-14 | injection\_show | mipicsi-stl | cat故障注入文件系统节点的时候的调用 |  |
| Comp-FU-006-15 | injection\_store | mipicsi-stl | 向故障注入文件系统节点echo值的时候的调用 |  |
| Comp-FU-006-16 | mipicsi\_stl\_init | mipicsi-stl | stl初始化 |  |
| Comp-FU-006-17 | mipicsi\_stl\_deinit | mipicsi-stl | stl注销 |  |
| Comp-FU-006-18 | eliminate\_jitter | mipicsi-stl | 事件消抖，降低事件上报频率 |  |
| Comp-FU-006-19 | mipicsi\_stl\_preinit | mipicsi-stl | 绑定stl ops |  |
| Comp-FU-006-20 | mipicsi\_stl\_injection\_test | mipicsi-stl | 向只读寄存器写0 |  |
| Comp-FU-006-21 | mipicsi\_stl\_inject\_test | mipicsi-stl | 统一故障注入接口 |  |
| Comp-FU-007-01 | phystl\_show | tztek\_phy\_stl | 故障注入sysfs节点：cat read的时候的函数入口 |  |
| Comp-FU-007-02 | phystl\_store | tztek\_phy\_stl | 故障注入sysfs节点：echo write的时候的函数入口 |  |
| Comp-FU-007-03 | phystl\_sysfs\_node\_destroy | tztek\_phy\_stl | phy stl 注销sysfs节点 |  |
| Comp-FU-007-04 | diag\_record\_event\_occurr\_time | tztek\_phy\_stl | 获取时间戳 |  |
| Comp-FU-007-05 | phy\_diag\_send\_event | tztek\_phy\_stl | 诊断事件发送 |  |
| Comp-FU-007-06 | store\_warning | tztek\_phy\_stl | 填充并打印payload，发送诊断事件 |  |
| Comp-FU-007-07 | phy\_fusa\_polling\_event | tztek\_phy\_stl | 周期性诊断入口 |  |
| Comp-FU-007-08 | phy\_irq\_handle | tztek\_phy\_stl | 中断处理函数：phy-stl |  |
| Comp-FU-007-09 | phy\_wq\_func | tztek\_phy\_stl | 异步工作队列：基于sm检查相关寄存器，必要时进行错误上报 |  |
| Comp-FU-007-10 | phy\_irq\_init | tztek\_phy\_stl | 创建phy-stl中断，使能功能安全机制 |  |
| Comp-FU-007-11 | phy\_diag\_info\_init | tztek\_phy\_stl | 事件结构体绑定stl |  |
| Comp-FU-007-12 | phy\_fusa\_init | tztek\_phy\_stl | phy fusa 注册初始化 |  |
| Comp-FU-007-13 | phy\_fusa\_deinit | tztek\_phy\_stl | phy fusa注销 |  |
| Comp-FU-007-14 | phystl\_sysfs\_node\_create | tztek\_phy\_stl | 故障注入sysfs节点创建 |  |
| Comp-FU-007-15 | phy\_fusa\_setup\_ops | tztek\_phy\_stl | 绑定stl ops |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

## 软件组件ASIL分配(不适用)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **函数名** | **函数运行的Core** | **函数运行的OS-Application** | **函数运行的OS-Application-Task** | **ASIL** | **备注** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## 组件动态设计

描述3.1功能分配后，3.2的架构如何使用

### 功能 001: ADC\_DIAG\_SWC

|  |  |
| --- | --- |
| **功能001** | **功能** |
| ADC\_DIAG\_SWC | 通过诊断驱动接口，及时上报驱动链路上出现的问题 |
|  | |

### 功能002:xxx

|  |  |
| --- | --- |
| **功能002** | **功能** |
| 操作概要 | 软件操作概要 |
| 软件单元级  函数调用图/功能流程图/状态机/数据流 | |

# 资源消耗目标的实现策略

## 为满足RAM资源消耗要求

1. 限制全局以及静态变量的使用

## 为满足flash资源消耗要求

1. 功能类似的模块在一个函数内实现，通过入参的形式来区分具体的功能，以减少代码量来满足flash资源消耗的要求

# 组件接口

## 5.1外部接口

与其他软件组件的接口. 描述细节，请使用“软件组件接口说明（独立模板）”

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **接口id** | **实现组件** | **接口名称** | **接口描述** | **from** | **to** | **接口原型** | **参数定义** | **参数名称** | **参数类型** | **有效值范围** | **单位** | **分辨率** |
| interface-201 | Module-001 | 注册执行接口 | 代码定义了一个宏，用于创建并启动一个内核线程，它接受线程函数、数据、线程名称等参数，并在创建成功后唤醒线程。 |  |  | kthread\_run(threadfn, data, namefmt, ...) | void \*threadfn,  u8 data,  u8 \*namefmt | threadfn | viod\* | 非null | 无 | 无 |
|  |  | data | u8 | 任意值 | 无 | 无 |
|  |  | namefmt | u8 \* | 非null | 无 | 无 |
| interface-202 | Module-001 | 注册执行接口 | 初始化自旋锁函数 |  |  | raw\_spin\_lock\_init(raw\_spinlock\_t) | typedef struct raw\_spinlock {  arch\_spinlock\_t raw\_lock;  #ifdef CONFIG\_GENERIC\_LOCKBREAK  unsigned int break\_lock;  #endif  #ifdef CONFIG\_DEBUG\_SPINLOCK  unsigned int magic, owner\_cpu;  void \*owner;  #endif  #ifdef CONFIG\_DEBUG\_LOCK\_ALLOC  struct lockdep\_map dep\_map;  #endif  } raw\_spinlock\_t; | raw\_spinlock\_t | strucct | 非null | 无 | 无 |

## 5.2内部接口

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **接口id** | **实现组件** | **接口名称** | **接口描述** | **from** | **to** | **接口原型** | **参数定义** | **参数名称** | **参数类型** | **有效值范围** | **单位** | **分辨率** |
| interface-201 | Module-001 | 注册执行接口 | 初始化diag\_core结构体的链表成员，及自旋锁成员 |  |  | static void diag\_lock\_init(void) | void | 无 | 无 | 无 | 无 | 无 |
| interface-202 | Module-001 | 注册执行接口 | 初始化存储和管理限流状态的相关信息 |  |  | static void diag\_print\_ratelimit\_init(void) | typedef struct raw\_spinlock {  arch\_spinlock\_t raw\_lock;  #ifdef CONFIG\_GENERIC\_LOCKBREAK  unsigned int break\_lock;  #endif  #ifdef CONFIG\_DEBUG\_SPINLOCK  unsigned int magic, owner\_cpu;  void \*owner;  #endif  #ifdef CONFIG\_DEBUG\_LOCK\_ALLOC  struct lockdep\_map dep\_map;  #endif  } raw\_spinlock\_t; | raw\_spinlock\_t | struct raw\_spinlock\* | 非null | 无 | 无 |
| interface-203 | Module-001 | 注册执行接口 | 创建一个新的线程来执行诊断轮询服务 |  |  | static int32\_t diag\_polling\_check\_service\_init(void) | void | 无 | 无 | 无 | 无 | 无 |
| interface-204 | Module-001 | 注册执行接口 | 初始化诊断资源，diag\_core结构体 |  |  | static int32\_t diag\_resource\_init(void) | void | 无 | 无 | 无 | 无 | 无 |
| interface-205 | Module-001 | 注册执行接口 | 释放诊断资源 |  |  | static void diag\_resource\_deinit(void) | void | 无 | 无 | 无 | 无 | 无 |
| interface-206 | Module-001 | 注册执行接口 | 程序退出时清理字符设备的资源 |  |  | static void diag\_chrdev\_deinit(struct diag \*diag) | struct diag{  struct device \*dev;  /\* list for all modules \*/  struct list\_head module\_list;  raw\_spinlock\_t module\_list\_spinlock;  /\* three priority level list \*/  struct list\_head low\_prio\_list;  struct list\_head mid\_prio\_list;  struct list\_head hig\_prio\_list;  raw\_spinlock\_t low\_spinlock;  raw\_spinlock\_t mid\_spinlock;  raw\_spinlock\_t hig\_spinlock;  struct work\_struct diag\_work;  struct diag\_module\_event \*module\_event;  struct list\_head available\_list;  raw\_spinlock\_t available\_spinlock;  struct diag\_netlink \*netlink;  struct diag\_chr\_dev \*chr\_dev;  struct diag\_record\_module\_register\_info diag\_record;  int32\_t diag\_polling\_check\_switch;  int32\_t send\_event\_cnt;  int32\_t handle\_event\_cnt;  } | diag | diag\* | 非null | 无 | 无 |
| interface-207 | Module-001 | 注册执行接口 | 释放网络链接相关的资源，包括关闭套接字描述符和销毁互斥锁 |  |  | static void diag\_netlink\_deinit(struct diag \*diag) | 同interface-206 | diag | diag\* | 非null | 无 | 无 |
| interface-208 | Module-001 | 注册执行接口 | 根据给定的ID返回相应的模块名称 |  |  | const uint8\_t \* fm\_name(uint16\_t module\_id) | typedef \_\_uint16\_t uint16\_t | module\_id | uint16\_t | 0<=module\_id<=ModuleIdMax | bity | 4 |
| interface-209 | Module-001 | 注册执行接口 | 检测diag\_register\_info参数是否满足的条件 |  |  | static int32\_t diag\_register\_arg\_check(const struct diag\_register\_info \*register\_info) | struct diag\_register\_info {  uint16\_t module\_id;  struct diag\_event\_id\_handle event\_handle[REGISTER\_EVENT\_ID\_MAX];  uint16\_t event\_cnt; }; | register\_info | diag\_register\_info\* | 非null | 无 | 无 |
| interface-210 | Module-001 | 注册执行接口 | 检查诊断模块的注册情况，包括模块ID和事件ID的重复注册检查，以及将新注册的事件信息添加到已注册模块中 |  |  | static int32\_t diag\_check\_module\_register(const struct diag\_register\_info \*register\_info, struct diag\_module \*module) | 参数1：struct diag\_register\_info {  uint16\_t module\_id;  struct diag\_event\_id\_handle event\_handle[REGISTER\_EVENT\_ID\_MAX];  uint16\_t event\_cnt; } | register\_info | diag\_register\_info \* |  |  |  |
| 参数2：struct diag\_module {  uint16\_t module\_id;  struct diag\_event\_id event\_id[REGISTER\_EVENT\_ID\_MAX];  uint8\_t event\_cnt;  uint8\_t module\_status;  struct list\_head list;} | module | diag\_module \* |  |  |  |
| interface-211 | Module-001 | 注册执行接口 | 根据模块id，检查诊断模块的注册情况，包括模块ID和事件ID的重复注册检查，以及将新注册的事件信息添加到已注册模块中 |  |  | static void diag\_register\_record(uint16\_t module\_id) | uint16\_t | module\_id | uint16\_t |  |  |  |
| interface-212 | Module-001 | 注册执行接口 | 检测事件是否满足条件，满足设置事件最后状态和最后发送时间 |  |  | static int32\_t diag\_check\_send\_condition(struct diag\_event\_id \*event\_id, struct diag\_event \*event) | 参数1：struct diag\_event\_id {  struct diag\_event\_id\_handle id\_handle;  uint8\_t last\_sta; //event last status  uint32\_t last\_snd\_time; //event last send time  } | event\_id | diag\_event\_id \* |  |  |  |
| 参数2：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event | diag\_event \* |  |  |  |
| interface-213 | Module-001 | 注册执行接口 | 检查和处理传入的事件结构体是否合法，如果合法则根据一定条件进行进一步处理 |  |  | static int32\_t diag\_send\_arg\_check(struct diag\_event \*event, int8\_t \*register\_state) | 参数1：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event | diag\_event \* |  |  |  |
| 参数2：int8\_t | register\_state | int8\_t \* |  |  |  |
| interface-214 | Module-001 | 注册执行接口 | 控制轮训诊断服务的开关状态 |  |  | int32\_t diag\_polling\_check\_service\_switch(int32\_t switch\_flag) | int32\_t | switch\_flag | int32\_t |  |  |  |
| interface-215 | Module-001 | 注册执行接口 | 控制轮训诊断服务的开关关闭 |  |  | static void diag\_polling\_check\_service\_deinit(void) |  |  |  |  |  |  |
| interface-216 | Module-001 | 注册执行接口 | 检查给定module\_id是否已经在记录中注册，如果注册了则返回0，如果未注册则返回-1 |  |  | static int diag\_register\_check\_record(uint16\_t module\_id) | uint16\_t | module\_id | uint16\_t |  |  |  |
| interface-217 | Module-001 | 注册执行接口 | 诊断模块列表中查找特定事件，如果找到则注册该事件，并返回对应的模块指针和事件索引。如果未找到匹配的模块或事件，则返回NULL |  |  | struct diag\_module \*diag\_module\_event\_register\_or\_not(struct diag\_event \*event, int32\_t \*event\_idx, uint8\_t status) | 参数1：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event | diag\_event \* |  |  |  |
| 参数2：int32\_t | event\_idx | int32\_t \* |  |  |  |
| 参数3：uint8\_t | status | uint8\_t |  |  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| interface-218 | Module-001 | 注册执行接口 | 打印诊断事件调试信息 |  |  | static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag) | 参数1：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  } | event | diag\_event \* |  |  |  |
| 参数2：int32\_t | open\_flag | int32\_t |  |  |  |
| interface-219 | Module-001 | 注册执行接口 | 打印事件data函数 |  |  | static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag) | 参数1：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  } | event | diag\_event \* |  |  |  |
| 参数2：int32\_t | open\_flag | int32\_t |  |  |  |
| interface-220 | Module-001 | 注册执行接口 | 将新的注册诊断模块添加到模块列表中 |  |  | static int32\_t diag\_register\_module\_to\_list(const struct diag\_register\_info \*register\_info) | struct diag\_register\_info {  uint16\_t module\_id;  struct diag\_event\_id\_handle event\_handle[REGISTER\_EVENT\_ID\_MAX];  uint16\_t event\_cnt;  }; | register\_info | diag\_register\_info \* |  |  |  |
| interface-221 | Module-001 | 注册执行接口 | 事件结构体 event 插入到级队列中 |  |  | static int32\_t diag\_insert\_to\_priority\_list(struct diag\_module\_event \*module\_event, struct diag\_event \*event) | 参数1：struct diag\_module\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code;  uint8\_t event\_sta;  uint8\_t event\_level;  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;  uint8\_t when;  uint8\_t ifenv;  struct list\_head list;  } | module\_event | diag\_module\_event |  |  |  |
| 参数2：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  } | event | diag\_event \* |  |  |  |
| interface-222 | Module-001 | 注册执行接口 | 事件结构体 event 插入到不同优先级队列中，并根据事件的优先级进行分类和处理 |  |  | static int32\_t diag\_module\_event\_add\_queue(struct diag\_event \*event) | struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event | diag\_event \* |  |  |  |
| interface-223 | Module-001 | 注册执行接口 | 遍历一个链表，处理每个元素的信息，并将处理完的元素移动到另一个链表module\_event |  |  | static void handle\_and\_move(const struct list\_head \*prio\_list, raw\_spinlock\_t \*lock) | 参数1：struct list\_head {  struct list\_head \*next, \*prev;  }; | prio\_list | const struct list\_head \* |  |  |  |
| 参数2：typedef struct raw\_spinlock {  arch\_spinlock\_t raw\_lock;  #ifdef CONFIG\_GENERIC\_LOCKBREAK  unsigned int break\_lock;  #endif  #ifdef CONFIG\_DEBUG\_SPINLOCK  unsigned int magic, owner\_cpu;  void \*owner;  #endif  #ifdef CONFIG\_DEBUG\_LOCK\_ALLOC  struct lockdep\_map dep\_map;  #endif  } raw\_spinlock\_t; | lock | raw\_spinlock\_t \* |  |  |  |
| interface-224 | Module-001 | 注册执行接口 | 通过传入的诊断事件，去调用与之匹配的回调函数 |  |  | static int32\_t diag\_handle\_cb(struct diag\_event \*event, uint32\_t errcode, uint32\_t critical) | 参数1：struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event： | struct diag\_event \* |  |  |  |
| 参数2：uint32\_t | errcode | uint32\_t |  |  |  |
| 参数3：uint32\_t | critical | uint32\_t |  |  |  |
| interface-225 | Module-001 | 注册执行接口 | 给模块事件分优先级 |  |  | static void diag\_work\_handler(struct work\_struct \*work) | struct work\_struct {  atomic\_long\_t data;  struct list\_head entry;  work\_func\_t func;  #ifdef CONFIG\_LOCKDEP  struct lockdep\_map lockdep\_map;  #endif  } | work | struct work\_struct \* |  |  |  |
| interface-226 | Module-001 | 注册执行接口 | 遍历模块列表中的每个模块，然后遍历每个模块的事件列表，如果事件的类型是EventIdPoll且有回调函数，就执行回调函数并传递相应参数 |  |  | static void diag\_execute\_polling\_event(void) |  |  |  |  |  |  |
| interface-227 | Module-001 | 注册执行接口 | 创建线程，每3s调用diag\_execute\_polling\_event |  |  | static int32\_t diag\_polling\_case\_thread(void \*arg) |  |  |  |  |  |  |
| interface-228 | Module-001 | 注册执行接口 | 检查某个诊断驱动程序是否已经初始化完成 |  |  | int32\_t diag\_is\_ready(void) |  |  |  |  |  |  |
| interface-229 | Module-001 | 注册执行接口 | 获取模块注册的状态 |  |  | int8\_t diagnose\_get\_register\_state(uint16\_t module\_id) | uint16\_t | module\_id | uint16\_t |  |  |  |
| interface-230 | Module-001 | 注册执行接口 | 检查参数有效，检查模块是否注册，没有就注册，将模块添加链表中，记录注册 |  |  | int32\_t diagnose\_register(const struct diag\_register\_info \*register\_info) | struct diag\_register\_info {  uint16\_t module\_id;  struct diag\_event\_id\_handle event\_handle[REGISTER\_EVENT\_ID\_MAX];  uint16\_t event\_cnt;  }; | register\_info | diag\_register\_info \* |  |  |  |
| interface-231 | Module-001 | 注册执行接口 | 调用检查事件对应模块是否注册，对事件参数检查，调用去执行回调函数，显示事件信息，将事件添加队列调schedule\_work 函数安排一个工作项，以便稍后 |  |  | int32\_t diagnose\_send\_event(struct diag\_event \*event) | struct diag\_event {  uint16\_t module\_id;  uint16\_t event\_id;  uint16\_t err\_code; // x9 define error code  uint8\_t event\_prio; //event priority  uint8\_t event\_sta; //event status  uint32\_t sec; // record occurrence\_time of the event  uint16\_t msec;  uint8\_t payload[ENV\_PAYLOAD\_SIZE];  uint8\_t env\_len;// every owner must check the value of env\_len !  uint8\_t when;  }; | event | struct diag\_event \* |  |  |  |
| interface-232 | Module-001 | 注册执行接口 | 检查诊断驱动程序的初始化状态和链表中的节点数目 |  |  | void diagnose\_get\_idle\_node\_number(uint16\_t \*number) |  |  |  |  |  |  |
| interface-233 | Module-001 | 注册执行接口 | 诊断驱动调用函数，初始化诊断资源 |  |  | static int diag\_probe(struct platform\_device \*pdev) | struct platform\_device {  const char \*name;  int id;  bool id\_auto;  struct device dev;  u32 num\_resources;  struct resource \*resource;  const struct platform\_device\_id \*id\_entry;  char \*driver\_override; /\* Driver name to force a match \*/  /\* MFD cell pointer \*/  struct mfd\_cell \*mfd\_cell;  /\* arch specific additions \*/  struct pdev\_archdata archdata;  }; | pdev | struct platform\_device\* |  |  |  |
| interface-234 | Module-001 | 注册执行接口 | 诊断驱动卸载调用函数，释放诊断资源 |  |  | static int diag\_remove(struct platform\_device \*pdev) | struct platform\_device {  const char \*name;  int id;  bool id\_auto;  struct device dev;  u32 num\_resources;  struct resource \*resource;  const struct platform\_device\_id \*id\_entry;  char \*driver\_override; /\* Driver name to force a match \*/  /\* MFD cell pointer \*/  struct mfd\_cell \*mfd\_cell;  /\* arch specific additions \*/  struct pdev\_archdata archdata;  }; | pdev | struct platform\_device\* |  |  |  |

# 一般定义

## 标识符预处理

本节确定了组件的所有宏定义

|  |  |  |
| --- | --- | --- |
| **名称** | 范围 | 描述 |
| diag\_printk\_ratelimited(\_rs, fmt, ...) | ({ \  if (\_\_ratelimit(&\_rs)) \  printk(fmt, ##\_\_VA\_ARGS\_\_); \  }) | 内核中实现有限速率的打印功能 |
| diag\_err\_ratelimit(\_rs, fmt, ...) | diag\_printk\_ratelimited(\_rs, KERN\_ERR pr\_fmt(fmt), ##\_\_VA\_ARGS\_\_) | 错误级别的消息格式 |
| diag\_warn\_ratelimit(\_rs, fmt, ...) | diag\_printk\_ratelimited(\_rs, KERN\_WARNING pr\_fmt(fmt), ##\_\_VA\_ARGS\_\_) | 警告级别的消息格式 |
| diag\_info\_ratelimit(\_rs, fmt, ...) | diag\_printk\_ratelimited(\_rs, KERN\_INFO pr\_fmt(fmt), ##\_\_VA\_ARGS\_\_) | 信息级别的消息格式 |
| DIAG\_SUCC | 0 | 诊断成功标识 |
| DIAG\_EINVAL | -1 | 诊断参数无效标识 |
| DIAG\_RESOURCE\_NO\_READY | -2 | 诊断资源没有准备成功标识 |
| DIAG\_NO\_REGISTER | -3 | 诊断模块没有注册标识 |
| DIAG\_MEM\_FAIL | -4 | 内存超出标识 |
| DIAG\_UNREGISTER\_FAIL | -5 | 诊断模块注销失败标识 |
| DIAG\_TIME\_INTERNEL | -6 | 诊断超时错误 |
| DIAG\_ADD\_LIST\_FAIL | -7 | 诊断模块添加链表错误 |
| DIAG\_REPEAT\_REGISTER | -8 | 事件添加链表错误 |
| EVENT\_ELEMENT\_ALL | 200 | 事件总数200 |
| EVENT\_DEBUG\_ELEMENT | 50 | 事件调试总数50 |
| DIAG\_MAX\_REPEAT\_NUM | 20 | 诊断最大的重复数量 |
| DAIG\_CHECK\_MODULE\_MAX\_ID | 252 | 诊断模块最大ID |
| DAIG\_POLLING\_CHECK\_TIME | (3\*HZ) | 诊断轮询检测的时间 |
| DAIG\_REGISTER\_OK | 1 | 诊断模块已经登记 |
| DAIG\_REGISTER\_NOT\_OK | 0 | 诊断模没有登记 |
| POLLING\_CHECK\_SERVICE\_OPEN | 1 | 轮询服务开启标志 |
| POLLING\_CHECK\_SERVICE\_CLOSE | 0 | 轮询服务关闭标志 |
| DEFAULT\_DIAG\_RATELIMIT\_INTERVAL | (5\*HZ) |  |
| DEFAULT\_DIAG\_RATELIMIT\_BURST | 500 |  |
| OPEN\_PRINTK | 1 | 事件数据开启打印 |
| CLOSE\_PRINTK | 0 | 事件数据关闭打印 |
| DIAG\_DRIVER\_OK | 1 | 诊断驱动加载成功 |
| DIAG\_DRIVER\_NOT\_OK | 0 | 诊断驱动加载失败 |

## 类型重定义

本节标识组件的所有结构体

|  |  |  |
| --- | --- | --- |
| **名称** | <类型名称> | |
| **类型** | <允许的条目：“枚举”、“结构”、“引用”（指针）a类型、integer数据类型> | |
| **成员/范围** | <法律价值范围> | <含义，单元，等> |
| **描述** | < #typedef的描述> | |

## 全局变量

本节确定了组件中使用的所有全局变量。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **名称** | **类型** | **描述** | | **初始值** | |
| 1 | diag\_drive\_init\_flay | atomic\_t | | 诊断驱动加载是否成功标识 | |  |
| 2 | diag\_core | struct diag\* | | 诊断模块，事件数据的集成 | |  |
| 3 | m\_name[ModuleIdMax] | uinit\_8 | | 模块名字集合 | | {" ","Diag","RTC",  "TickA","TickR",  "i2c9","i2c16",  "eth0", "spi5",  "sdhci1", "mipicsi0", "csi0",  "cambus","dc0",  "dsi0", "lcmbus",  "ulinkpcie",  "pcie2","gpio4",  "uart8","uart11",  "uart12","uart13",  "uart14","mobox",  "wdt5","wdt6",  "a55", } |

## 静态变量

本节确定了组件中使用的所有静态变量。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **序号** | **名称** | **类型** | **描述** | | **初始值** | |
| 1 | diag\_drive\_init\_flay | atomic\_t | | 诊断驱动加载是否成功标识 |  |
| 2 | diag\_core | struct diag\* | | 诊断模块，事件数据的集成 |  |
| 3 | print\_ratelimit  [ModuleDiag\_kernelMax] | ratelimit\_state | | 速率限制的状态结构体集合 | {0} |
| 4 | m\_name[ModuleIdMax] | uinit\_8 | | 模块名字集合 | {" ","Diag","RTC",  "TickA","TickR",  "i2c9","i2c16",  "eth0", "spi5",  "sdhci1", "mipicsi0", "csi0",  "cambus","dc0",  "dsi0", "lcmbus",  "ulinkpcie",  "pcie2","gpio4",  "uart8","uart11",  "uart12","uart13",  "uart14","mobox",  "wdt5","wdt6",  "a55", } |

# 功能单元详细信息

## 功能函数: static void diag\_lock\_init(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_lock\_init | |
| **激活时机** | diag驱动匹配成功 | |
| **输入** | void | 无 |
| **输出** | void | 无 |
| **返回值** | void | 无 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 初始化diag\_core结构体的链表成员，及自旋锁成员 | |

## 功能函数:static void diag\_print\_ratelimit\_init(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_print\_ratelimit\_init | |
| **激活时机** | diag驱动匹配成功 | |
| **输入** | void | 无 |
| **输出** | void | 无 |
| **返回值** | void | 无 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 初始化存储和管理限流状态的相关信息 | |

## 功能函数:static int32\_t diag\_polling\_check\_service\_init(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_polling\_check\_service\_init | |
| **激活时机** | diag驱动匹配成功 | |
| **输入** | void | 无 |
| **输出** | None |  |
| **返回值** | int32\_t | 0：success  -1：fail |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 创建一个新的线程来执行诊断轮询服务 | |

## 功能函数:static int32\_t diag\_resource\_init(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_resource\_init | |
| **激活时机** | diag驱动匹配成功 | |
| **输入** | void | 无 |
| **输出** | None |  |
| **返回值** | int32\_t | 0：初始化成功  -1：初始化失败 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 初始化诊断资源，diag\_core结构体 | |

## 功能函数:static void diag\_resource\_deinit(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_resource\_deinit | |
| **激活时机** | diag驱动移除 | |
| **输入** | void | None |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 释放诊断资源 | |

## 功能函数:static void diag\_chrdev\_deinit(struct diag \*diag)

|  |  |  |
| --- | --- | --- |
| **功能名称** | void diag\_chrdev\_deinit | |
| **激活时机** | diag驱动移除 | |
| **输入** | diag | 诊断结构体指针 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 程序退出时清理字符设备的资源 | |

## 功能函数:static void diag\_netlink\_deinit(struct diag \*diag)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_netlint\_deinit | |
| **激活时机** | diag驱动移除 | |
| **输入** | diag | netlink诊断结构体指针 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 释放网络链接相关的资源，包括关闭套接字描述符和销毁互斥锁 | |

## 功能函数:const uint8\_t \* fm\_name(uint16\_t module\_id)

|  |  |  |
| --- | --- | --- |
| **功能名称** | fm\_name | |
| **激活时机** |  | |
| **输入** | uint16\_t | 模块id |
| **输出** | None |  |
| **返回值** | uint8\_t\* | 模块名称 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 根据给定的ID返回相应的模块名称 | |

## 功能函数:static int32\_t diag\_register\_arg\_check(const struct diag\_register\_info \*register\_info)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_register\_arg\_check | |
| **激活时机** | 注册诊断模块 | |
| **输入** | register\_info | 模块的数据结构体指针 |
| **输出** | None |  |
| **返回值** | int32\_t | 0:模块数据正常  DIAG\_EINVAL：模块数据异常 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检测diag\_register\_info参数是否满足的条件 | |

## 功能函数:static int32\_t diag\_check\_module\_register(const struct diag\_register\_info \*register\_info, struct diag\_module \*module)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_check\_module\_register | |
| **激活时机** | 注册诊断模块 | |
| **输入** | register\_info | 需要注册的诊断事件，模块数据结构体指针 |
| diag\_module | 已经注册的诊断模块数据结构体指针 |
| **输出** | None |  |
| **返回值** | int32\_t | DIAG\_NO\_REGISTER:该诊断模块还未注册  DIAG\_REPEAD\_REGISTER:该诊断模块的事件已经注册  DIAG\_SUCC：该诊断模块的事件注册成功 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检查诊断模块的注册情况，包括模块ID和事件ID的重复注册检查，以及将新注册的事件信息添加到已注册模块中 | |

## 功能函数:static void diag\_register\_record(uint16\_t module\_id)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_register\_record | |
| **激活时机** | 注册诊断模块 | |
| **输入** | uint16\_t | 模块ID |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 根据模块id，检查诊断模块的注册情况，包括模块ID和事件ID的重复注册检查，以及将新注册的事件信息添加到已注册模块中 | |

## 功能函数:static int32\_t diag\_check\_send\_condition(struct diag\_event\_id \*event\_id, struct diag\_event \*event)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_check\_send\_condition | |
| **激活时机** | 诊断事件上报 | |
| **输入** | event\_id | 传输的事件数据 |
| event | 注册的事件数据 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检测事件是否满足条件，满足设置事件最后状态和最后发送时间 | |

## 功能函数:static int32\_t diag\_send\_arg\_check(struct diag\_event \*event, int8\_t \*register\_state)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_send\_arg\_check | |
| **激活时机** | 诊断事件上报 | |
| **输入** | event | 诊断事件结构体 |
| register\_state | 登记状态 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检查和处理传入的事件结构体是否合法，如果合法则根据一定条件进行进一步处理 | |

## 功能函数:int32\_t diag\_polling\_check\_service\_switch(int32\_t switch\_flag)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_polling\_check\_service\_switch | |
| **激活时机** | 轮询诊断关闭，开启时刻 | |
| **输入** | int32\_t | 0：关闭轮询诊断  1：开启轮询诊断 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 控制轮训诊断服务的开关状态 | |

## 功能函数:static void diag\_polling\_check\_service\_deinit(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_polling\_check\_service\_deinit | |
| **激活时机** | 诊断轮询模式注销 | |
| **输入** | void | None |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 控制轮训诊断服务的开关关闭 | |

## 功能函数:static int diag\_register\_check\_record(uint16\_t module\_id)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_regster\_check\_record | |
| **激活时机** | 诊断模块注册时 | |
| **输入** | uint16\_t | 模块ID |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检查给定module\_id是否已经在记录中注册，如果注册了则返回0，如果未注册则返回-1 | |

## 功能函数:struct diag\_module \*diag\_module\_event\_register\_or\_not(struct diag\_event \*event, int32\_t \*event\_idx, uint8\_t status)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_module\_event\_register\_or\_not | |
| **激活时机** | 诊断模块注册时 | |
| **输入** | event | 诊断事件 |
| event\_idx | 事件数组下标 |
| status | 模块状态 |
| **输出** | None |  |
| **返回值** | diag\_module | 诊断模块 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 诊断模块列表中查找特定事件，如果找到则注册该事件，并返回对应的模块指针和事件索引。如果未找到匹配的模块或事件，则返回NULL | |

## 功能函数:static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_show\_event\_info | |
| **激活时机** | 事件信息打印时 | |
| **输入** | event | 诊断事件结构体 |
| open\_flag | 打印标识 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 打印诊断事件调试信息 | |

## 功能函数:static void diag\_show\_event\_info(struct diag\_event \*event, int32\_t open\_flag)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_show\_event\_info | |
| **激活时机** | 事件信息打印时 | |
| **输入** | event | 诊断事件结构体 |
| open\_flag | 打印标识 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 诊断接口eth是否有接受或发送错误的数据包 | |

## 功能函数:static int32\_t diag\_register\_module\_to\_list(const struct diag\_register\_info \*register\_info)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_register\_module\_to\_list | |
| **激活时机** | 诊断模块登记时 | |
| **输入** | register\_info | 诊断事件数据结构体 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 将新的注册诊断模块添加到模块列表中 | |

## 功能函数:static int32\_t diag\_insert\_to\_priority\_list(struct diag\_module\_event \*module\_event, struct diag\_event \*event)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_insert\_to\_priority\_list | |
| **激活时机** | 诊断事件传递时 | |
| **输入** | module\_event | 模块注册的事件 |
| event | 传输的事件 |
| **输出** | None |  |
| **返回值** | int32\_t | -1:添加到priority链表失败  0:添加到priority链表成功 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 事件结构体 event 插入到级队列中 | |

## 功能函数:static int32\_t diag\_module\_event\_add\_queue(struct diag\_event \*event)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_module\_event\_add\_queue | |
| **激活时机** | 事件登记时 | |
| **输入** | event | 诊断事件结构体 |
| **输出** | None |  |
| **返回值** | int32\_t | -1:添加到priority链表失败  0:添加到priority链表成功 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 事件结构体 event 插入到不同优先级队列中，并根据事件的优先级进行分类和处理 | |

## 功能函数:static void handle\_and\_move(const struct list\_head \*prio\_list, raw\_spinlock\_t \*lock)

|  |  |  |
| --- | --- | --- |
| **功能名称** | handle\_and\_move | |
| **激活时机** | 处理诊断事件 | |
| **输入** | prio\_list | 事件储存的链表 |
| lock | 自旋锁 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 遍历一个链表，处理每个元素的信息，并将处理完的元素移动到另一个链表module\_event | |

## 功能函数:static int32\_t diag\_handle\_cb(struct diag\_event \*event, uint32\_t errcode, uint32\_t critical)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_handle\_cb | |
| **激活时机** | 事件信息打印时 | |
| **输入** | event | 诊断事件结构体 |
| errcode | 错误码 |
| critical | 是否为重要错误 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 通过传入的诊断事件，去调用与之匹配的回调函数 | |

## 功能函数:static void diag\_work\_handler(struct work\_struct \*work)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_work\_handler | |
| **激活时机** | 诊断事件处理时 | |
| **输入** | work | 工作流 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 给模块事件分优先级 | |

## 功能函数:static void diag\_execute\_polling\_event(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_execute\_polling\_event | |
| **激活时机** | 处理诊断事件时 | |
| **输入** | void | null |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 遍历模块列表中的每个模块，然后遍历每个模块的事件列表，如果事件的类型是EventIdPoll且有回调函数，就执行回调函数并传递相应参数 | |

## 功能函数:static int32\_t diag\_polling\_case\_thread(void \*arg)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_polling\_case\_thread | |
| **激活时机** | 处理诊断事件时 | |
| **输入** | arg | None |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 创建线程，每3s调用diag\_execute\_polling\_event | |

## 功能函数:int32\_t diag\_is\_ready(void)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_is\_ready | |
| **激活时机** | 处理诊断事件时 | |
| **输入** | void | None |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检查某个诊断驱动程序是否已经初始化完成 | |

## 功能函数:int8\_t diagnose\_get\_register\_state(uint16\_t module\_id)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diagnose\_get\_register\_state | |
| **激活时机** | 获取诊断模块注册状态时 | |
| **输入** | module\_id | 模块id |
| **输出** | None |  |
| **返回值** | int8\_t | DIAG\_EINVAL:参数无效  DIAG\_RESOURCE\_NO\_READY：资源没有准备好  0：DiagEventStaSuccess  1：DiagEventStaFail  -1:其他错误 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 获取模块注册的状态 | |

## 功能函数:int32\_t diagnose\_register(const struct diag\_register\_info \*register\_info)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diagnose\_register | |
| **激活时机** | 诊断模块注册时 | |
| **输入** | register\_info | 诊断模块数据 |
| **输出** | None |  |
| **返回值** | int32\_t | DIAG\_EINVAL:参数无效  DIAG\_RESOURCE\_NO\_READY：资源没有准备好  DIAG\_REPEAT\_REGISTER:重复注册  DIAG\_SUCC:注册成功 |
| **程序流** | R | |
| **算法** | None | |
| **解释** | 检查参数有效，检查模块是否注册，没有就注册，将模块添加链表中，记录注册的模块 | |

## 功能函数:int32\_t diagnose\_send\_event(struct diag\_event \*event)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diagnose\_send\_event | |
| **激活时机** | 处理诊断模块事件时 | |
| **输入** | event | 诊断事件结构体 |
| **输出** | None |  |
| **返回值** | int32\_t | DIAG\_EINVAL:参数无效  DIAG\_ADD\_LIST\_FAIL：添加到链表失败  DIAG\_SUCC:事件传递成功  -1：失败 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 调用检查事件对应模块是否注册，对事件参数检查，调用去执行回调函数，显示事件信息，将事件添加队列调schedule\_work 函数安排一个工作项，以便稍后异步处理事件 | |

## 功能函数:void diagnose\_get\_idle\_node\_number(uint16\_t \*number)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diagnose\_get\_idle\_node\_number | |
| **激活时机** |  | |
| **输入** | number | 返回的空闲的节点数 |
| **输出** | None |  |
| **返回值** | void | None |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 检查诊断驱动程序的初始化状态和链表中的节点数目 | |

## 功能函数:static int diag\_probe(struct platform\_device \*pdev)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_probe | |
| **激活时机** | 诊断驱动compatible时 | |
| **输入** | pdev | 平台设备结构体 |
| **输出** | None |  |
| **返回值** | int | -1：失败  ENOMEN:内存分配失败  0：正常返回 |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 诊断驱动调用函数，初始化诊断资源 | |

## 功能函数:static int diag\_remove(struct platform\_device \*pdev)

|  |  |  |
| --- | --- | --- |
| **功能名称** | diag\_remove | |
| **激活时机** | 卸载诊断时 | |
| **输入** | pdev | 平台设备结构体 |
| **输出** | None |  |
| **返回值** | int32\_t |  |
| **程序流** |  | |
| **算法** | None | |
| **解释** | 诊断驱动卸载调用函数，释放诊断资源 | |

# 其他

评估详细设计。

从互操作性、交互、关键性、技术复杂性、风险和可测试性方面对软件详细设计进行评估。