模块间通信设计文档

# 主机端：通信状态机



# 从机端：通信状态机



# 三层设计的文件结构



# 驱动层接口

## 串口1 API

|  |  |
| --- | --- |
| 串口1API | |
| usart1\_config(); | 串口1配置 |
| usart1\_close(); | 关闭串口1 |
| usart1\_clear\_receive\_count(); | 清空接收数据计数 |
| usart1\_sub\_timeout(); | 超时计数递减函数（由定时器周期调用） |
| usart1\_get\_timeout(); | 获取当前超时计数值 |
| usart1\_judge\_timout(); | 判断超时计数时间到 |
| usart1\_reset\_timeout(); | 复位超时计数值 |
| usart1\_get\_send\_buf(); | 获取发送缓冲区 |
| usart1\_clear\_receive\_data\_count(); | 清空接收数据计数 |
| usart1\_get\_receive\_data\_count(); | 获取接收数据计数 |
| usart1\_get\_receive\_data(); | 获取接收数据地址 |
| usart1\_get\_send\_status(); | 获取串口的发送状态（读发送完成标记） |
| usart1\_send\_data(); | 发送数据 |
| usart1\_resend\_data(); | 重发数据 |
| get\_usart1\_receive\_over\_flag(); | 获取接收完成标记 |
| clear\_usart1\_receive\_over\_flag(); | 清空接收完成标记 |

## usart1\_config()

描述

配置串口1

原型

void usart1\_config(uint32\_t baud\_rate);

|  |  |
| --- | --- |
| 参数 | 描述 |
| baud\_rate | 波特率（115200 与模块保持一致） |

示例

usart1\_config(115200);

## usart1\_close()

描述

关闭串口1

原型

void usart1\_close(void);

示例

usart1\_close();

## usart1\_clear\_receive\_count()

描述

清空串口1接收数据计数

原型

void usart1\_clear\_receive\_count(void);

示例

usart1\_clear\_receive\_count ();

## usart1\_sub\_timeout()

描述

串口1接收超时计数减1并判断是否减到0，如果减到0就将接收完成标记置1，返回最新的超时计数值。

原型

uint32\_t usart1\_sub\_timeout(void);

其他信息

该函数设计为定时器1ms调用1次

示例

usart1\_sub\_timeout ();

## usart1\_get\_timeout()

描述

获取超时计数值单位ms

原型

uint32\_t usart1\_get\_timeout(void);

示例

uint32\_t timeout;

timeout = usart1\_sub\_timeout ();

## usart1\_judge\_timout()

描述

判断串口1超时计数时间到，时间到将接收完成标记置1

原型

void usart1\_judge\_timout(void);

示例

usart1\_judge\_timout ();

相关主题

与usart1\_get\_timeout()，usart1\_sub\_timeout ()配合使用相当于usart1\_sub\_timeout()

## usart1\_reset\_timeout()

描述

复位接收超时计数值，在接收中断中进行

原型

void usart1\_reset\_timeout(void);

示例

usart1\_reset\_timeout ();

相关主题

与usart1\_get\_timeout()，usart1\_sub\_timeout ()，usart1\_sub\_timeout()

## usart1\_get\_send\_buf()

描述

获取串口1的发送缓冲区

原型

uint8\_t \* usart1\_get\_send\_buf(void);

示例

uint8\_t \*buf;

buf = usart1\_get\_send\_buf ();

## usart1\_clear\_receive\_data\_count()

描述

清空接收数据计数

原型

void usart1\_clear\_receive\_data\_count(void);

示例

usart1\_clear\_receive\_data\_count();

## usart1\_get\_receive\_data\_count()

描述

获取串口1接收数据计数

原型

uint32\_t usart1\_get\_receive\_data\_count(void);

示例

uint32\_t count;

count = usart1\_get\_receive\_data\_count ();

## usart1\_get\_receive\_data()

描述

获取串口1接收数据缓冲区地址

原型

uint8\_t\* usart1\_get\_receive\_data(void);

示例

uint8\_t \*data;

data = usart1\_get\_receive\_data ();

## usart1\_get\_send\_status()

描述

获取串口1发送状态,发送完成后发送状态标记被置1

原型

uint8\_t usart1\_get\_send\_status(void);

示例

uint8\_t status;

status = usart1\_get\_send\_status ();

## usart1\_send\_data()

描述

串口1的发送数据函数

原型

void usart1\_send\_data(uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| data | 要发送的数据流地址 |
| len | 要发送的数据字节数 |

示例

usart1\_send\_data(data, len);

## usart1\_resend\_data()

描述

串口1的重发最后一次发送的数据

原型

void usart1\_resend\_data(void);

示例

usart1\_resend\_data();

## get\_usart1\_receive\_over\_flag()

描述

获取串口1接收完成标记

原型

uint8\_t get\_usart1\_receive\_over\_flag(void);

示例

uint8\_t over\_flag;

over\_flag = get\_usart1\_receive\_over\_flag ();

## clear\_usart1\_receive\_over\_flag()

描述

清空串口1接收完成标记

原型

void clear\_usart1\_receive\_over\_flag(void);

示例

clear\_usart1\_receive\_over\_flag ();

## 串口2 API

|  |  |
| --- | --- |
| 串口2API | |
| usart2\_config(); | 串口2配置 |
| usart2\_close(); | 关闭串口2 |
| usart2\_clear\_receive\_count(); | 清空接收数据计数 |
| usart2\_sub\_timeout(); | 超时计数递减函数（由定时器周期调用） |
| usart2\_get\_timeout(); | 获取当前超时计数值 |
| usart2\_judge\_timout(); | 判断超时计数时间到 |
| usart2\_reset\_timeout(); | 复位超时计数值 |
| usart2\_get\_send\_buf(); | 获取发送缓冲区 |
| usart2\_clear\_receive\_data\_count(); | 清空接收数据计数 |
| usart2\_get\_receive\_data\_count(); | 获取接收数据计数 |
| usart2\_get\_receive\_data(); | 获取接收数据地址 |
| usart2\_get\_send\_status(); | 获取串口的发送状态（读发送完成标记） |
| usart2\_send\_data(); | 发送数据 |
| usart2\_resend\_data(); | 重发数据 |
| get\_usart2\_receive\_over\_flag(); | 获取接收完成标记 |
| clear\_usart2\_receive\_over\_flag(); | 清空接收完成标记 |

## usart2\_config()

描述

配置串口2

原型

void usart2\_config(uint32\_t baud\_rate);

|  |  |
| --- | --- |
| 参数 | 描述 |
| baud\_rate | 波特率（115200 与模块保持一致） |

示例

usart2\_config(115200);

## usart2\_close()

描述

关闭串口2

原型

void usart2\_close(void);

示例

usart2\_close();

## usart2\_clear\_receive\_count()

描述

清空串口2接收数据计数

原型

void usart2\_clear\_receive\_count(void);

示例

usart2\_clear\_receive\_count ();

## usart2\_sub\_timeout()

描述

串口2接收超时计数减1并判断是否减到0，如果减到0就将接收完成标记置1，返回最新的超时计数值。

原型

uint32\_t usart2\_sub\_timeout(void);

其他信息

该函数设计为定时器1ms调用1次

示例

usart2\_sub\_timeout ();

## usart2\_get\_timeout()

描述

获取超时计数值单位ms

原型

uint32\_t usart2\_get\_timeout(void);

示例

uint32\_t timeout;

timeout = usart2\_sub\_timeout ();

## usart2\_judge\_timout()

描述

判断串口2超时计数时间到，时间到将接收完成标记置1

原型

void usart2\_judge\_timout(void);

示例

usart2\_judge\_timout ();

相关主题

与usart2\_get\_timeout()，usart2\_sub\_timeout ()配合使用相当于usart2\_sub\_timeout()

## usart2\_reset\_timeout()

描述

复位接收超时计数值，在接收中断中进行

原型

void usart2\_reset\_timeout(void);

示例

usart2\_reset\_timeout ();

相关主题

与usart2\_get\_timeout()，usart2\_sub\_timeout ()，usart2\_sub\_timeout()

## usart2\_get\_send\_buf()

描述

获取串口2的发送缓冲区

原型

uint8\_t \* usart2\_get\_send\_buf(void);

示例

uint8\_t \*buf;

buf = usart2\_get\_send\_buf ();

## usart2\_clear\_receive\_data\_count()

描述

清空接收数据计数

原型

void usart2\_clear\_receive\_data\_count(void);

示例

usart2\_clear\_receive\_data\_count();

## usart2\_get\_receive\_data\_count()

描述

获取串口2接收数据计数

原型

uint32\_t usart2\_get\_receive\_data\_count(void);

示例

uint32\_t count;

count = usart2\_get\_receive\_data\_count ();

## usart2\_get\_receive\_data()

描述

获取串口2接收数据缓冲区地址

原型

uint8\_t\* usart2\_get\_receive\_data(void);

示例

uint8\_t \*data;

data = usart2\_get\_receive\_data ();

## usart2\_get\_send\_status()

描述

获取串口2发送状态,发送完成后发送状态标记被置1

原型

uint8\_t usart2\_get\_send\_status(void);

示例

uint8\_t status;

status = usart2\_get\_send\_status ();

## usart2\_send\_data()

描述

串口2的发送数据函数

原型

void usart2\_send\_data(uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| data | 要发送的数据流地址 |
| len | 要发送的数据字节数 |

示例

usart2\_send\_data(data, len);

## usart2\_resend\_data()

描述

串口2的重发最后一次发送的数据

原型

void usart2\_resend\_data(void);

示例

usart2\_resend\_data();

## get\_usart2\_receive\_over\_flag()

描述

获取串口2接收完成标记

原型

uint8\_t get\_usart2\_receive\_over\_flag(void);

示例

uint8\_t over\_flag;

over\_flag = get\_usart2\_receive\_over\_flag ();

## clear\_usart2\_receive\_over\_flag()

描述

清空串口2接收完成标记

原型

void clear\_usart2\_receive\_over\_flag(void);

示例

clear\_usart2\_receive\_over\_flag ();

## 串口3 API

|  |  |
| --- | --- |
| 串口3API | |
| usart3\_config(); | 串口3配置 |
| usart3\_close(); | 关闭串口3 |
| usart3\_clear\_receive\_count(); | 清空接收数据计数 |
| usart3\_sub\_timeout(); | 超时计数递减函数（由定时器周期调用） |
| usart3\_get\_timeout(); | 获取当前超时计数值 |
| usart3\_judge\_timout(); | 判断超时计数时间到 |
| usart3\_reset\_timeout(); | 复位超时计数值 |
| usart3\_get\_send\_buf(); | 获取发送缓冲区 |
| usart3\_clear\_receive\_data\_count(); | 清空接收数据计数 |
| usart3\_get\_receive\_data\_count(); | 获取接收数据计数 |
| usart3\_get\_receive\_data(); | 获取接收数据地址 |
| usart3\_get\_send\_status(); | 获取串口的发送状态（读发送完成标记） |
| usart3\_send\_data(); | 发送数据 |
| usart3\_resend\_data(); | 重发数据 |
| get\_usart3\_receive\_over\_flag(); | 获取接收完成标记 |
| clear\_usart3\_receive\_over\_flag(); | 清空接收完成标记 |

## usart3\_config()

描述

配置串口3

原型

void usart3\_config(uint32\_t baud\_rate);

|  |  |
| --- | --- |
| 参数 | 描述 |
| baud\_rate | 波特率（115200 与模块保持一致） |

示例

usart3\_config(115200);

## usart3\_close()

描述

关闭串口3

原型

void usart3\_close(void);

示例

usart3\_close();

## usart3\_clear\_receive\_count()

描述

清空串口3接收数据计数

原型

void usart3\_clear\_receive\_count(void);

示例

usart3\_clear\_receive\_count ();

## usart3\_sub\_timeout()

描述

串口3接收超时计数减3并判断是否减到0，如果减到0就将接收完成标记置1，返回最新的超时计数值。

原型

uint32\_t usart3\_sub\_timeout(void);

其他信息

该函数设计为定时器1ms调用1次

示例

usart3\_sub\_timeout ();

## usart3\_get\_timeout()

描述

获取超时计数值单位ms

原型

uint32\_t usart3\_get\_timeout(void);

示例

uint32\_t timeout;

timeout = usart3\_sub\_timeout ();

## usart3\_judge\_timout()

描述

判断串口3超时计数时间到，时间到将接收完成标记置1

原型

void usart3\_judge\_timout(void);

示例

usart3\_judge\_timout ();

相关主题

与usart3\_get\_timeout()，usart3\_sub\_timeout ()配合使用相当于usart3\_sub\_timeout()

## usart3\_reset\_timeout()

描述

复位接收超时计数值，在接收中断中进行

原型

void usart3\_reset\_timeout(void);

示例

usart3\_reset\_timeout ();

相关主题

与usart3\_get\_timeout()，usart3\_sub\_timeout ()，usart3\_sub\_timeout()

## usart3\_get\_send\_buf()

描述

获取串口3的发送缓冲区

原型

uint8\_t \* usart3\_get\_send\_buf(void);

示例

uint8\_t \*buf;

buf = usart3\_get\_send\_buf ();

## usart3\_clear\_receive\_data\_count()

描述

清空接收数据计数

原型

void usart3\_clear\_receive\_data\_count(void);

示例

usart3\_clear\_receive\_data\_count();

## usart3\_get\_receive\_data\_count()

描述

获取串口3接收数据计数

原型

uint32\_t usart3\_get\_receive\_data\_count(void);

示例

uint32\_t count;

count = usart3\_get\_receive\_data\_count ();

## usart3\_get\_receive\_data()

描述

获取串口3接收数据缓冲区地址

原型

uint8\_t\* usart3\_get\_receive\_data(void);

示例

uint8\_t \*data;

data = usart3\_get\_receive\_data ();

## usart3\_get\_send\_status()

描述

获取串口3发送状态,发送完成后发送状态标记被置1

原型

uint8\_t usart3\_get\_send\_status(void);

示例

uint8\_t status;

status = usart3\_get\_send\_status ();

## usart3\_send\_data()

描述

串口3的发送数据函数

原型

void usart3\_send\_data(uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| data | 要发送的数据流地址 |
| len | 要发送的数据字节数 |

示例

usart3\_send\_data(data, len);

## usart3\_resend\_data()

描述

串口3的重发最后一次发送的数据

原型

void usart3\_resend\_data(void);

示例

usart3\_resend\_data();

## get\_usart3\_receive\_over\_flag()

描述

获取串口3接收完成标记

原型

uint8\_t get\_usart3\_receive\_over\_flag(void);

示例

uint8\_t over\_flag;

over\_flag = get\_usart3\_receive\_over\_flag ();

## clear\_usart3\_receive\_over\_flag()

描述

清空串口3接收完成标记

原型

void clear\_usart3\_receive\_over\_flag(void);

示例

clear\_usart3\_receive\_over\_flag ();

## 串口4 API

|  |  |
| --- | --- |
| 串口4API | |
| uart4\_config(); | 串口4配置 |
| uart4\_close(); | 关闭串口4 |
| uart4\_clear\_receive\_count(); | 清空接收数据计数 |
| uart4\_sub\_timeout(); | 超时计数递减函数（由定时器周期调用） |
| uart4\_get\_timeout(); | 获取当前超时计数值 |
| uart4\_judge\_timout(); | 判断超时计数时间到 |
| uart4\_reset\_timeout(); | 复位超时计数值 |
| uart4\_get\_send\_buf(); | 获取发送缓冲区 |
| uart4\_clear\_receive\_data\_count(); | 清空接收数据计数 |
| uart4\_get\_receive\_data\_count(); | 获取接收数据计数 |
| uart4\_get\_receive\_data(); | 获取接收数据地址 |
| uart4\_get\_send\_status(); | 获取串口的发送状态（读发送完成标记） |
| uart4\_send\_data(); | 发送数据 |
| uart4\_resend\_data(); | 重发数据 |
| get\_uart4\_receive\_over\_flag(); | 获取接收完成标记 |
| clear\_uart4\_receive\_over\_flag(); | 清空接收完成标记 |

## uart4\_config()

描述

配置串口4

原型

void uart4\_config(uint32\_t baud\_rate);

|  |  |
| --- | --- |
| 参数 | 描述 |
| baud\_rate | 波特率（115200 与模块保持一致） |

示例

uart4\_config(115200);

## uart4\_close()

描述

关闭串口4

原型

void uart4\_close(void);

示例

uart4\_close();

## uart4\_clear\_receive\_count()

描述

清空串口4接收数据计数

原型

void uart4\_clear\_receive\_count(void);

示例

uart4\_clear\_receive\_count ();

## uart4\_sub\_timeout()

描述

串口4接收超时计数减1并判断是否减到0，如果减到0就将接收完成标记置1，返回最新的超时计数值。

原型

uint32\_t uart4\_sub\_timeout(void);

其他信息

该函数设计为定时器1ms调用1次

示例

uart4\_sub\_timeout ();

## uart4\_get\_timeout()

描述

获取超时计数值单位ms

原型

uint32\_t uart4\_get\_timeout(void);

示例

uint32\_t timeout;

timeout = uart4\_sub\_timeout ();

## uart4\_judge\_timout()

描述

判断串口4超时计数时间到，时间到将接收完成标记置1

原型

void uart4\_judge\_timout(void);

示例

uart4\_judge\_timout ();

相关主题

与uart4\_get\_timeout()，uart4\_sub\_timeout ()配合使用相当于uart4\_sub\_timeout()

## uart4\_reset\_timeout()

描述

复位接收超时计数值，在接收中断中进行

原型

void uart4\_reset\_timeout(void);

示例

uart4\_reset\_timeout ();

相关主题

与uart4\_get\_timeout()，uart4\_sub\_timeout ()，uart4\_sub\_timeout()

## uart4\_get\_send\_buf()

描述

获取串口4的发送缓冲区

原型

uint8\_t \* uart4\_get\_send\_buf(void);

示例

uint8\_t \*buf;

buf = uart4\_get\_send\_buf ();

## uart4\_clear\_receive\_data\_count()

描述

清空接收数据计数

原型

void uart4\_clear\_receive\_data\_count(void);

示例

uart4\_clear\_receive\_data\_count();

## uart4\_get\_receive\_data\_count()

描述

获取串口4接收数据计数

原型

uint32\_t uart4\_get\_receive\_data\_count(void);

示例

uint32\_t count;

count = uart4\_get\_receive\_data\_count ();

## uart4\_get\_receive\_data()

描述

获取串口4接收数据缓冲区地址

原型

uint8\_t\* uart4\_get\_receive\_data(void);

示例

uint8\_t \*data;

data = uart4\_get\_receive\_data ();

## uart4\_get\_send\_status()

描述

获取串口4发送状态,发送完成后发送状态标记被置1

原型

uint8\_t uart4\_get\_send\_status(void);

示例

uint8\_t status;

status = uart4\_get\_send\_status ();

## uart4\_send\_data()

描述

串口4的发送数据函数

原型

void uart4\_send\_data(uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| data | 要发送的数据流地址 |
| len | 要发送的数据字节数 |

示例

uart4\_send\_data(data, len);

## uart4\_resend\_data()

描述

串口4的重发最后一次发送的数据

原型

void uart4\_resend\_data(void);

示例

uart4\_resend\_data();

## get\_uart4\_receive\_over\_flag()

描述

获取串口4接收完成标记

原型

uint8\_t get\_uart4\_receive\_over\_flag(void);

示例

uint8\_t over\_flag;

over\_flag = get\_uart4\_receive\_over\_flag ();

## clear\_uart4\_receive\_over\_flag()

描述

清空串口4接收完成标记

原型

void clear\_uart4\_receive\_over\_flag(void);

示例

clear\_uart4\_receive\_over\_flag ();

## 定时器3API

|  |  |
| --- | --- |
| 定时器3API | |
| tim3\_init (); | 定时器3初始化 |
| register\_tim3\_server\_fun (); | 注册定时器3服务函数 |

## tim3\_init ()

描述

初始化定时器3

原型

void tim3\_init(uint16\_t arr,uint16\_t psc);

|  |  |
| --- | --- |
| 参数 | 描述 |
| arr | 定时器预装值 |
| psc | 时钟分频值(定时器时钟是84M STM32F407) |

示例

tim3\_init(10 - 1, 8400 - 1);//10 \* 0.1ms = 1ms //定时器3初始化

## register\_tim3\_server\_fun ()

描述

定时器3服务函数注册，最多可以注册TIM3\_MAX\_SERVER\_FUN个服务函数，可根据实际使用情况进行调整

原型

uint8\_t register\_tim3\_server\_fun(void (\*fun)(void));

|  |  |
| --- | --- |
| 参数 | 描述 |
| fun | 定时器服务函数 |

示例

tim3\_init(10 - 1, 8400 - 1);//10 \* 0.1ms = 1ms //定时器3初始化

res = register\_tim3\_server\_fun(server\_fun);

# 应用层接口

|  |  |
| --- | --- |
| 多路模块管理API | |
| init\_module\_manage\_env(); | 初始化模块管理环境 |
| module\_comm\_task(); | 模块通信任务 |
| clear\_module\_inf(); | 清空模块信息，重新获取模块信息时使用 |
| com\_comm\_is\_idle(); | 判断串口是否在空闲状态 |
| wait\_com\_comm\_idle(); | 等待串口进入空闲状态 |
| com\_module\_connect(); | 发送模块连接指令，第一种形式定义 |
| com\_module\_set\_road\_num(); | 发送设置模块路编号指令 |
| send\_module\_connect(); | 发送模块连接指令，第二种形式定义 |

## init\_module\_manage\_env()

描述

初始化模块管理环境,该函数只能被调用1次，内部实现有动态分配的内存

原型

void init\_module\_manage\_env(void);

示例

init\_module\_manage\_env();

## module\_comm\_task()

描述

模块通信使用的任务，维护各个串口状态机的运行。由用户层任务直接调用。

原型

void module\_comm\_task(void);

示例

module\_comm\_task ();

## clear\_module\_inf()

描述

清空模块信息，当重新获取模块信息时需要先清空模块信息

原型

void clear\_module\_inf(void);

示例

clear\_module\_inf ();

## com\_comm\_is\_idle()

描述

通信串口的状态机是否处于空闲状态，返回CS\_TRUE表示空闲状态，返回CS\_FALSE表示不在空闲状态

原型

CS\_BOOL com\_comm\_is\_idle(MODULE\_ADDR\_T addr);

|  |  |
| --- | --- |
| 参数 | 描述 |
| addr | 模块地址  0-15 串口1的模块地址范围  16-31 串口2的模块地址范围  32-47 串口3的模块地址范围48-63 串口4的模块地址范围 |

示例

CS\_BOOL bool\_flag;

bool\_flag = com\_comm\_is\_idle (1);

## wait\_com\_comm\_idle()

描述

等待模块所对应的串口状态机进入空闲状态

原型

void wait\_com\_comm\_idle(MODULE\_ADDR\_T addr);

|  |  |
| --- | --- |
| 参数 | 描述 |
| addr | 模块地址  0-15 串口1的模块地址范围  16-31 串口2的模块地址范围  32-47 串口3的模块地址范围48-63 串口4的模块地址范围 |

示例

wait\_com\_comm\_idle (1);

## com\_module\_connect()

描述

发送模块连接指令，这是第一种形式的函数定义

原型

CS\_ERR com\_module\_connect(MODULE\_ADDR\_T addr);

|  |  |
| --- | --- |
| 参数 | 描述 |
| addr | 模块地址  0-15 串口1的模块地址范围  16-31 串口2的模块地址范围  32-47 串口3的模块地址范围48-63 串口4的模块地址范围 |

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_ERR\_COM\_BUSY; | 串口状态机正在忙状态，放弃发送 |
| CS\_ERR\_SEND\_SUCCESS | 发送成功 |

示例

CS\_ERR err;

err = com\_module\_connect (1);

## com\_module\_set\_road\_num()

描述

发送设置模块路编号指令，告诉每个模块它在系统中的通信地址。

原型

CS\_ERR com\_module\_set\_road\_num(MODULE\_ADDR\_T addr, uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| addr | 模块地址  0-15 串口1的模块地址范围  16-31 串口2的模块地址范围  32-47 串口3的模块地址范围48-63 串口4的模块地址范围 |
| data | 指令携带的数据缓冲区地址 |
| len | 指令携带数据长度 |

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_ERR\_COM\_BUSY; | 串口状态机正在忙状态，放弃发送 |
| CS\_ERR\_SEND\_SUCCESS | 发送成功 |

示例

CS\_ERR err;

ROAD\_NUM\_T road\_num = 1;

err = com\_module\_set\_road\_num (1, &road\_num, sizeof(road\_num));

## send\_module\_connect()

描述

发送模块连接指令，这是第二种形式的函数定义

原型

CS\_ERR send\_module\_connect(MODULE\_ADDR\_T addr, uint8\_t \*data, uint32\_t len);

|  |  |
| --- | --- |
| 参数 | 描述 |
| addr | 模块地址  0-15 串口1的模块地址范围  16-31 串口2的模块地址范围  32-47 串口3的模块地址范围48-63 串口4的模块地址范围 |
| data | 指令携带的数据缓冲区地址 |
| len | 指令携带数据长度 |

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_ERR\_COM\_BUSY; | 串口状态机正在忙状态，放弃发送 |
| CS\_ERR\_SEND\_SUCCESS | 发送成功 |

示例

CS\_ERR err;

ROAD\_NUM\_T road\_num = 1;

err = send\_module\_connect (1, NULL, 0);

# 模块扫描功能API

|  |  |
| --- | --- |
| 模块扫描功能API | |
| com1\_scan\_is\_over() | 判断第一路串口扫描是否结束 |
| com2\_scan\_is\_over() | 判断第二路串口扫描是否结束 |
| com3\_scan\_is\_over() | 判断第三路串口扫描是否结束 |
| com4\_scan\_is\_over() | 判断第四路串口扫描是否结束 |
| get\_com1\_scan\_addr() | 获取当前第一路串口正在扫描的模块地址 |
| get\_com2\_scan\_addr() | 获取当前第二路串口正在扫描的模块地址 |
| get\_com3\_scan\_addr() | 获取当前第三路串口正在扫描的模块地址 |
| get\_com4\_scan\_addr() | 获取当前第四路串口正在扫描的模块地址 |
| scan\_all\_module() | 扫描所有模块，循环调用 |
| stop\_scan\_all\_module() | 停止扫描所有模块 |
| start\_scan\_all\_module() | 启动扫描所有模块 |

# com1\_scan\_is\_over()

描述

第一路串口是否扫描结束判断

原型

CS\_BOOL com1\_scan\_is\_over(void);

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_TRUE | 扫描结束 |
| CS\_FALSE | 扫描未结束 |

示例

CS\_BOOL flag;

flag = com1\_scan\_is\_over ();

# com2\_scan\_is\_over()

描述

第二路串口是否扫描结束判断

原型

CS\_BOOL com2\_scan\_is\_over(void);

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_TRUE | 扫描结束 |
| CS\_FALSE | 扫描未结束 |

示例

CS\_BOOL flag;

flag = com2\_scan\_is\_over ();

# com3\_scan\_is\_over()

描述

第三路串口是否扫描结束判断

原型

CS\_BOOL com3\_scan\_is\_over(void);

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_TRUE | 扫描结束 |
| CS\_FALSE | 扫描未结束 |

示例

CS\_BOOL flag;

flag = com3\_scan\_is\_over ();

# com4\_scan\_is\_over()

描述

第四路串口是否扫描结束判断

原型

CS\_BOOL com4\_scan\_is\_over(void);

返回值

|  |  |
| --- | --- |
| 值 | 描述 |
| CS\_TRUE | 扫描结束 |
| CS\_FALSE | 扫描未结束 |

示例

CS\_BOOL flag;

flag = com4\_scan\_is\_over ();

# get\_com1\_scan\_addr()

描述

获取第一路串口正在扫描的模块地址

原型

MODULE\_ADDR\_T get\_com1\_scan\_addr(void);

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| MODULE\_ADDR\_T | 第一路串口正扫描模块的地址(1 - 15) |

示例

MODULE\_ADDR\_T addr;

addr = get\_com1\_scan\_addr ();

# get\_com2\_scan\_addr()

描述

获取第二路串口正在扫描的模块地址

原型

MODULE\_ADDR\_T get\_com2\_scan\_addr(void);

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| MODULE\_ADDR\_T | 第二路串口正扫描模块的地址(1 - 15) |

示例

MODULE\_ADDR\_T addr;

addr = get\_com2\_scan\_addr ();

# get\_com3\_scan\_addr()

描述

获取第三路串口正在扫描的模块地址

原型

MODULE\_ADDR\_T get\_com3\_scan\_addr(void);

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| MODULE\_ADDR\_T | 第三路串口正扫描模块的地址(1 - 15) |

示例

MODULE\_ADDR\_T addr;

addr = get\_com3\_scan\_addr ();

# get\_com4\_scan\_addr()

描述

获取第二路串口正在扫描的模块地址

原型

MODULE\_ADDR\_T get\_com4\_scan\_addr(void);

返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| MODULE\_ADDR\_T | 第二路串口正扫描模块的地址(1 - 15) |

示例

MODULE\_ADDR\_T addr;

addr = get\_com4\_scan\_addr ();

# scan\_all\_module()

描述

扫描所有模块，由用户层循环调用

原型

void scan\_all\_module(void)

示例

scan\_all\_module ();

# stop\_scan\_all\_module()

描述

停止扫描所有模块

原型

void stop\_scan\_all\_module(void);

示例

stop\_scan\_all\_module ();

# start\_scan\_all\_module()

描述

停止扫描所有模块

原型

void start\_scan\_all\_module(void);

示例

start\_scan\_all\_module ();

extern CS\_BOOL com1\_send\_is\_over(void);

extern CS\_BOOL com2\_send\_is\_over(void);

extern CS\_BOOL com3\_send\_is\_over(void);

extern CS\_BOOL com4\_send\_is\_over(void);

extern CS\_BOOL all\_com\_send\_is\_over(void);

extern uint8\_t get\_com1\_send\_addr(void);

extern uint8\_t get\_com2\_send\_addr(void);

extern uint8\_t get\_com3\_send\_addr(void);

extern uint8\_t get\_com4\_send\_addr(void);

extern void send\_all\_module(uint8\_t \*data, uint32\_t len);

extern void stop\_send\_all\_module(void);

extern void start\_send\_all\_module(void);

extern void init\_send\_module\_connect(void);

extern void init\_send\_set\_road\_num(void);

extern void init\_send\_module\_connect(void);

extern CS\_ERR connect\_module(uint8\_t addr);

# 用户层运行流程