嵌入式C/C++代码规范

1. 文件组织

1.1 文件命名

- 源文件使用 `.c` (C语言) 或 `.cpp` (C++)
- 头文件使用 `.h` (C/C++通用) 或 `.hpp` (C++专用)
- 文件名使用小写字母和下划线： `uart_driver.c` ， `sensor_manager.h`
- 模块相关文件使用相同前缀： `spi_init.c` ， `spi_read.c` ， `spi_write.c`

1.2 头文件保护

```
#ifndef MODULE_NAME_H
#define MODULE_NAME_H

/* 头文件内容 */

#endif /* MODULE_NAME_H */
```

1.3 文件结构顺序

```c
/* 1. 文件头注释 */
/*******************************************************************************
 * @file    uart_driver.c
 * @brief   UART驱动程序实现
 * @author  张三
 * @date    2024-01-01
 * @version V1.0.0
 *******************************************************************************/

/* 2. 包含头文件 */
#include <stdint.h>         /* 标准库 */
#include "config.h"         /* 配置文件 */
#include "uart_driver.h"    /* 本模块头文件 */

/* 3. 宏定义 */
#define BUFFER_SIZE  256

/* 4. 类型定义 */
typedef struct {...} uart_config_t;

/* 5. 全局变量声明 */
extern uint32_t g_system_clock;

/* 6. 静态变量定义 */
static uint8_t s_rx_buffer[BUFFER_SIZE];

/* 7. 函数原型声明 */
static void uart_isr_handler(void);

/* 8. 函数实现 */
```

2. 命名规范

2.1 变量命名

```
/* 局部变量：小写字母+下划线 */
uint32_t byte_count;
char *p_buffer;

/* 全局变量：g_ 前缀 */
uint32_t g_system_tick;

/* 静态变量：s_ 前缀 */
static uint8_t s_uart_state;

/* 常量：大写字母+下划线 */
const uint32_t MAX_RETRY_COUNT = 3;

/* 指针变量：p_ 前缀（可选） */
char *p_data;
```

2.2 函数命名

```
/* 模块名_动作_对象 */
void uart_send_byte(uint8_t data);
bool spi_read_register(uint8_t reg_addr, uint8_t *p_value);

/* 获取/设置函数 */
uint32_t timer_get_count(void);
void timer_set_period(uint32_t period);

/* 初始化/反初始化 */
void module_init(void);
void module_deinit(void);
```

2.3 宏定义命名

```
/* 常量宏：全大写+下划线 */
#define UART_BAUDRATE_115200  115200
#define BUFFER_SIZE           256

/* 功能宏：全大写+下划线 */
#define MIN(a, b)  ((a) < (b) ? (a) : (b))
#define BIT(n)     (1UL << (n))

/* 寄存器定义 */
#define UART_BASE_ADDR  0x40004000
#define UART_CR        (*((volatile uint32_t *)(UART_BASE_ADDR + 0x00)))
```

2.4 类型定义命名

```
/* 结构体：_t 后缀 */
typedef struct {
    uint32_t baudrate;
    uint8_t  data_bits;
    uint8_t  stop_bits;
} uart_config_t;

/* 枚举：_e 后缀 */
typedef enum {
    UART_STATE_IDLE = 0,
    UART_STATE_BUSY,
    UART_STATE_ERROR
} uart_state_e;

/* 联合体：_u 后缀 */
typedef union {
    uint32_t word;
    uint8_t  bytes[4];
} data_converter_u;
```

3. 代码格式

3.1 缩进与空格

- 使用4个空格缩进，不使用Tab
- 运算符两边加空格
- 逗号后面加空格
- 分号后面加空格（for循环中）

```
/* 正确示例 */
for (i = 0; i < count; i++) {
    sum = sum + array[i];
}

/* 错误示例 */
for(i=0;i<count;i++){
    sum=sum+array[i];
}
```

3.2 大括号风格

```
/* 函数定义：大括号另起一行 */
void function_name(void)
{
    /* 函数体 */
}

/* 控制语句：大括号同一行 */
if (condition) {
    /* 代码块 */
} else {
    /* 代码块 */
}

/* 结构体定义 */
typedef struct {
    uint32_t field1;
    uint32_t field2;
} struct_name_t;
```

3.3 行宽限制

- 每行代码不超过80个字符
- 长表达式合理换行

```
/* 函数参数过多时换行 */
error_code = uart_configure(UART_INSTANCE_1,
                            BAUDRATE_115200,
                            DATA_BITS_8,
                            STOP_BITS_1,
                            PARITY_NONE);

/* 长条件表达式换行 */
if ((uart_state == UART_STATE_READY) &&
    (tx_buffer_count > 0) &&
    (dma_channel_available == true)) {
    /* 处理逻辑 */
}
```

4. 注释规范

4.1 函数注释

```
/**
 * @brief  发送数据通过UART
 * @param  p_data: 指向要发送数据的指针
 * @param  length: 要发送的字节数
 * @retval 0: 成功, -1: 失败
 * @note   此函数会阻塞直到所有数据发送完成
 */
int32_t uart_send_data(const uint8_t *p_data, uint32_t length)
{
    /* 函数实现 */
}
```

4.2 代码注释

```
/* 单行注释：解释下一行代码 */
timeout_count = TIMEOUT_VALUE;

/*
 * 多行注释：
 * 用于解释复杂的逻辑
 * 或算法实现步骤
 */

/* TODO: 添加错误处理 */
/* FIXME: 修复内存泄漏问题 */
/* NOTE: 此处使用忙等待是为了确保时序准确 */
```

5. 编程实践

5.1 变量初始化

```
/* 定义时初始化 */
uint32_t count = 0;
char *p_buffer = NULL;

/* 结构体初始化 */
uart_config_t config = {
    .baudrate = 115200,
    .data_bits = 8,
    .stop_bits = 1,
    .parity = PARITY_NONE
};
```

5.2 错误处理

```
/* 使用错误码 */
typedef enum {
    ERR_OK = 0,
    ERR_INVALID_PARAM = -1,
    ERR_TIMEOUT = -2,
    ERR_BUSY = -3
} error_code_e;

/* 错误检查示例 */
error_code_e uart_init(const uart_config_t *p_config)
{
    if (p_config == NULL) {
        return ERR_INVALID_PARAM;
    }

    /* 初始化代码 */

    return ERR_OK;
}
```

5.3 防御性编程

```c
/* 参数检查 */
void process_data(uint8_t *p_buffer, uint32_t size)
{
    /* 空指针检查 */
    if (p_buffer == NULL) {
        return;
    }

    /* 边界检查 */
    if (size > MAX_BUFFER_SIZE) {
        size = MAX_BUFFER_SIZE;
    }

    /* 处理逻辑 */
}

/* 断言使用（仅在调试版本） */
#ifdef DEBUG
    #define ASSERT(expr) do { \
        if (!(expr)) { \
            assert_failed(__FILE__, __LINE__); \
        } \
    } while(0)
#else
    #define ASSERT(expr) ((void)0)
#endif
```

5.4 内存管理

```
/* 动态内存分配检查 */
uint8_t *p_buffer = (uint8_t *)malloc(size);
if (p_buffer == NULL) {
    /* 错误处理 */
    return ERR_NO_MEMORY;
}

/* 使用完后释放 */
free(p_buffer);
p_buffer = NULL;  /* 避免野指针 */

/* 优先使用栈和静态分配 */
static uint8_t s_dma_buffer[DMA_BUFFER_SIZE];  /* 静态分配 */
uint8_t local_buffer[32];                      /* 栈分配 */
```

5.5 中断处理

```c
/* 中断服务函数：简短快速 */
void UART_IRQHandler(void)
{
    uint32_t status = UART->SR;

    if (status & UART_SR_RXNE) {
        /* 接收数据到缓冲区 */
        g_rx_buffer[g_rx_index++] = UART->DR;
        g_rx_flag = 1;  /* 设置标志，主循环处理 */
    }

    /* 清除中断标志 */
    UART->SR = 0;
}

/* 临界区保护 */
uint32_t enter_critical(void)
{
    uint32_t primask = __get_PRIMASK();
    __disable_irq();
    return primask;
}

void exit_critical(uint32_t primask)
{
    __set_PRIMASK(primask);
}
```

6. 嵌入式特定规范

6.1 寄存器访问

```
/* 使用volatile关键字 */
#define REG32(addr)  (*((volatile uint32_t *)(addr)))
#define REG16(addr)  (*((volatile uint16_t *)(addr)))
#define REG8(addr)   (*((volatile uint8_t *)(addr)))

/* 位操作宏 */
#define SET_BIT(REG, BIT)     ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)   ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)    ((REG) & (BIT))
#define TOGGLE_BIT(REG, BIT)  ((REG) ^= (BIT))
```

6.2 内存对齐

```
/* 结构体对齐 */
typedef struct {
    uint8_t  byte1;
    uint8_t  byte2;
    uint16_t word1;    /* 2字节对齐 */
    uint32_t dword1;   /* 4字节对齐 */
} __attribute__((packed)) packed_struct_t;

/* DMA缓冲区对齐 */
__attribute__((aligned(4))) uint8_t dma_buffer[256];
```

6.3 ROM优化

```
/* 常量数据放入ROM */
const uint8_t lookup_table[256] = {
    /* 查找表数据 */
};

/* 字符串常量 */
const char *const error_messages[] = {
    "Success",
    "Invalid parameter",
    "Timeout",
    "Buffer overflow"
};
```

6.4 功耗优化

```
/* 进入低功耗模式 */
void enter_low_power_mode(void)
{
    /* 关闭不需要的外设时钟 */
    RCC->APB1ENR &= ~(RCC_APB1ENR_TIM2EN | RCC_APB1ENR_TIM3EN);

    /* 配置GPIO为低功耗状态 */
    configure_gpio_for_low_power();

    /* 进入睡眠模式 */
    __WFI();
}
```

7. 代码审查要点

7.1 安全性检查

- 数组边界检查
- 指针有效性验证
- 整数溢出防护
- 栈溢出预防

7.2 可靠性检查

- 错误返回值处理
- 资源释放（内存、文件句柄等）
- 死锁避免
- 竞态条件防护

7.3 性能检查

- 循环优化
- 内存访问局部性
- 缓存友好的数据结构
- 避免不必要的复制

7.4 可维护性检查

- 代码可读性
- 注释完整性
- 模块化程度
- 接口清晰度

8. 版本控制规范

8.1 提交信息格式

```
[类型] 简短描述

详细说明（可选）

相关issue: #123
```

类型包括：

- feat: 新功能
- fix: 修复bug
- docs: 文档更新
- style: 代码格式调整
- refactor: 重构
- test: 测试相关

- chore: 构建过程或辅助工具的变动

8.2 分支管理

- master/main: 稳定版本
- develop: 开发分支
- feature/xxx: 功能分支
- bugfix/xxx: 修复分支
- release/x.x.x: 发布分支

9. 附录：常用缩写规范

缩写	全称	说明
addr	address	地址
buf/buff	buffer	缓冲区
cfg/config	configuration	配置
cnt	count	计数
ctrl	control	控制
dev	device	设备
err	error	错误
init	initialize	初始化
len	length	长度
max	maximum	最大值
min	minimum	最小值
msg	message	消息
num	number	数量
param	parameter	参数
ptr/p	pointer	指针
recv/rx	receive	接收

缩写	全称	说明
reg	register	寄存器
req	request	请求
resp/rsp	response	响应
ret/retval	return value	返回值
send/tx	transmit	发送
src	source	源
stat	status	状态
sync	synchronize	同步
temp/tmp	temporary	临时
val	value	值

10. 代码模板示例

10.1 模块头文件模板

```c
#ifndef MODULE_NAME_H
#define MODULE_NAME_H

#ifdef __cplusplus
extern "C" {
#endif

/*******************************************************************************
 * Includes
 ******************************************************************************/
#include <stdint.h>
#include <stdbool.h>

/*******************************************************************************
 * Defines
 ******************************************************************************/
#define MODULE_VERSION  "1.0.0"

/*******************************************************************************
 * Typedefs
 ******************************************************************************/
typedef struct {
    uint32_t param1;
    uint32_t param2;
} module_config_t;

/*******************************************************************************
 * Exported Functions
 ******************************************************************************/
int32_t module_init(const module_config_t *p_config);
int32_t module_deinit(void);
int32_t module_process(uint8_t *p_data, uint32_t length);

#ifdef __cplusplus
}
#endif
```

```
#endif /* MODULE_NAME_H */
```

10.2 模块源文件模板

```c
/*******************************************************************************
 * @file    module_name.c
 * @brief   模块功能简述
 * @author  作者
 * @date    日期
 * @version V1.0.0
 *******************************************************************************/

/*******************************************************************************
 * Includes
 *******************************************************************************/
#include "module_name.h"

/*******************************************************************************
 * Private Defines
 *******************************************************************************/
#define INTERNAL_BUFFER_SIZE  128

/*******************************************************************************
 * Private Types
 *******************************************************************************/
typedef struct {
    uint8_t  buffer[INTERNAL_BUFFER_SIZE];
    uint32_t index;
    bool     is_initialized;
} module_context_t;

/*******************************************************************************
 * Private Variables
 *******************************************************************************/
static module_context_t s_context = {0};

/*******************************************************************************
 * Private Function Prototypes
 *******************************************************************************/
static int32_t validate_config(const module_config_t *p_config);
static void process_internal(void);

/*******************************************************************************
 * Public Functions
 *******************************************************************************/
```

```c
/**
 * @brief  初始化模块
 * @param  p_config: 配置参数指针
 * @retval 0: 成功, <0: 错误码
 */
int32_t module_init(const module_config_t *p_config)
{
    int32_t ret;

    /* 参数检查 */
    ret = validate_config(p_config);
    if (ret != 0) {
        return ret;
    }

    /* 初始化上下文 */
    memset(&s_context, 0, sizeof(s_context));
    s_context.is_initialized = true;

    return 0;
}

/*****************************************************************************
 * Private Functions
 *****************************************************************************/

static int32_t validate_config(const module_config_t *p_config)
{
    if (p_config == NULL) {
        return -1;
    }

    /* 验证配置参数 */

    return 0;
}
```

注意事项：

1. 本规范可根据具体项目和团队需求进行调整

2. 使用自动化工具（如clang-format）确保代码格式一致性
3. 定期进行代码审查，确保规范得到遵守
4. 对于特定MCU平台，参考厂商提供的编程指南