WeilaiChip
R&D

WL818 SDK 框架规则

产品应用手册 V1.0.0 Date: 2026/3/12 蔚来芯研发部

摘要说明

类别	内容
关键词	WL818 SDK
摘 要	WL818 SDK 框架规则

修订历史

版本	修订人	时间	审核	备注
V1.0.0	李元鹏	2026/3/12		初次修订

目 录

1. 适用范围和参考资料

本文档规格仅应用于蔚来芯研发部，作为 WL818 SDK 的编写框架指导。

2. WL818 SDK 框架规则

2.1 WL818 SDK 分层架构

架构分为 hal 层和 driver 层，hal 层为直接暴露给客户的接口，格式参考如下：

```
/**
* @brief  MIPI TX发送命令
* @param  data_type: 数据类型
* @param  vc: 虚拟通道编号
* @param  cmd_count: 可变参数个数
* @param  ...: 可变参数
* @retval true-command发送正常；false-TX当前状态不能发送command
*/
bool hal_dsi_tx_ctrl_write_cmd(uint8_t data_type, uint8_t vc, uint8_t cmd_count, ...);

/**
* @brief  MIPI TX发送命令
* @param  data_type: 数据类型
* @param  vc: 虚拟通道编号
* @param  size: data个数
* @param  data: data数组
* @retval true-command发送正常；false-TX当前状态不能发送command
*/
bool hal_dsi_tx_ctrl_write_array_cmd(uint8_t data_type, uint8_t vc, uint8_t size, const uint8_t *data);

/**
* @brief  控制码片bist输出
* @param  bist_en: 使能bist
* @param  BIST_MODE: bist的模式
* @retval
*/
void hal_dsi_tx_ctrl_set_bist(bool bist_en, BIST_MODE bist_mode);

/**
* @brief  MIPI TX接收命令
* @param  data_type: 数据类型
* @param  vc: 虚拟通道编号
* @param  cmd: DCS指令
* @param  size: 读取数据长度
* @param  data: 数据存放地址
* @retval true-回读正常；false-当前状态不能回读
*/
bool hal_dsi_tx_ctrl_read_cmd(uint8_t data_type, uint8_t vc, uint8_t cmd, uint8_t size, uint8_t *data);
```

而其中的寄存器操作，统一封装到 driver 层，仅为内部使用，抽象寄存器操作增加代码可读性，格式参考如下：

```
/**
* @brief  控制mipi模块按照设定的配置开始工作
* @param  none
* @retval none
*/
static inline bool drv_mipi_tx_work()
{
    MIPI_TX_PORT0->tx_control_L = 0x03;
}

/**
* @brief  获取mipi tx busy状态
* @param  none
* @retval true: MIPI TX模块处理中，false:MIPI TX模块处理完成
*/
static inline bool drv_mipi_tx_get_busy()
{
    return TAU_BIT_CHECK_ZERO(MIPI_TX_PORT0->tx_state, 0);
}

/**
* @brief  获取mipi tx read的值
* @param  none
* @retval mipi读取的值
*/
static inline uint32_t drv_mipi_tx_get_read_result()
{
    return MIPI_TX_PORT0->rd_lp_rx_data;
}

/**
* @brief  控制mipi模块写数据
* @param  data: 数据buffer
* @param  cmd_count: 数据长度
* @param  tx_mode: 发送模式 1: HS 0:LP
* @retval none
*/
void drv_mipi_tx_write_data(const uint8_t *cmd, uint8_t cmd_count, TX_MODE tx_mode);

/**
* @brief  控制mipi模块清空lp read
* @param  none
* @retval none
*/
void drv_mipi_tx_lprd_clear();
```

2.2 hal 层封装规则

由于 hal 层接口直接被用户使用，所以需要对入口参数进行检测，参考如下

```c
/**
* @brief  MIPI TX接收命令
* @param  data_type: 数据类型
* @param  vc: 虚拟通道编号
* @param  cmd: DCS指令
* @param  size: 读取数据长度
* @param  data: 数据存放地址
* @retval true-回读正常；false-当前状态不能回读
*/
bool hal_dsi_tx_ctrl_read_cmd(uint8_t data_type, uint8_t vc, uint8_t cmd, uint8_t size, uint8_t *data)
{
    /*验证参数有效性*/
    if ((size < 1) || (NULL == data))
    {
        return false;
    }

    drv_mipi_tx_lprd_clear();
    drv_mipi_tx_set_max_size(size);
    delay_ms(100);
    drv_mipi_tx_read_data(cmd);

    uint32_t result = drv_mipi_tx_get_read_result();      You, 昨天 • Uncommitted changes
    for(int i = 0; i < size; i++)
    {
        data[i] = ((result >> i) & 0xFF);
    }

    return true;
}
```

2.3 driver 层封装规则

driver 层封装寄存器操作，让代码可读性更高，对于寄存器地址的抽象，如果是连续地址，可抽象成结构体，后续操作通过结构体指针进行操作，参考如下：

c > driver > include > C drv_mipi_tx.h > _MIPI_TX_PORT0_REG > tx_enable

```c
#ifndef __DRV_MIPI_TX_H__
#include "typedef.h"
#include <stdarg.h>
#include "la_config.h"

#ifndef MIPI_TX_PORT0_BASE
#define  MIPI_TX_PORT0_BASE              (0x80010000)
#endif

#define MIPI_TX_PORT0 ((MIPI_TX_PORT0_REG *)MIPI_TX_PORT0_BASE)

typedef struct _MIPI_TX_PORT0_REG
{
    __O  uint32_t       write_data      ;
    __IO uint8_t        tx_enable       ;
    __IO uint8_t        tx_control_L    ;
    __IO uint8_t        tx_control_H    ;
    __I  uint8_t        tx_state        ;
    __IO uint32_t       rd_lp_rx_data   ;
    __IO uint32_t       rd_lp_rx_ecc    ;
    __O  uint32_t       fifo_set        ;
    __I  uint32_t       ISR             ;
    __IO uint32_t       IMR             ;
    __O  uint32_t       ICR             ;

}MIPI_TX_PORT0_REG;

/*!
 * @brief MIPI发送模式
 * @note
 */
typedef enum _TX_MODE
{
    LP_MODE = 0,
    HS_MODE,
}TX_MODE;

/**
* @brief  向mipi模块发出写数据配置申请，后续才能向mipi模块配置需要写的数据
* @param  none
* @retval none
*/
static inline void drv_mipi_tx_write_configure_enable()
{
    MIPI_TX_PORT0->tx_enable = 0x01;
}
```

寄存器地址抽象

操作寄存器

对于一些高频操作，也可以宏定义的方式增加代码可读性，参考如下：

```c
#include "drv_mipi_tx.h"

/** \brief 设置mipi模块发送模式
 * tx_mode参考枚举TX_MODE，1为高速发送，0为低速发送
 */
#define SET_MIPI_MODE(tx_mode) (write_addr_UINT8(mipi_tx_lp_hs_mode, tx_mode))

/** \brief 设置mipi模块发送的数据
 * data为具体数据
 */
#define SET_MIPI_DATA(data) (write_addr_UINT32(mipi_tx_send_data, data))

/**
* @brief  控制mipi模块写数据
* @param  cmd: 数据buffer
* @param  cmd_count: 数据长度
* @param  tx_mode: 发送模式 1: HS 0:LP
* @retval none
*/
void drv_mipi_tx_write_data(const uint8_t *cmd, uint8_t cmd_count, TX_MODE tx_mode)
{
    if(cmd_count > 1)
    {
        SET_MIPI_MODE(tx_mode);

        /*配置mipi模块要写的数据*/
        drv_mipi_tx_write_configure_enable();
        SET_MIPI_DATA((cmd[0] << 24) + (cmd_count << 8) + 0x39 + (0x3 << 6));
        for (uint8_t i = 0; i < ((cmd_count - 1) >> 2); i++)
        {
            SET_MIPI_DATA((cmd[4 + (i << 2)] << 24) + (cmd[3 + (i << 2)] << 16) + (cmd[2 + (i << 2)] << 8) + (cmd[1 + (i << 2)]));
        }
        switch ((cmd_count - 1) % 4)
        {
        case 1:
            SET_MIPI_DATA(cmd[cmd_count - 1]);
            break;

        case 2:
            SET_MIPI_DATA((cmd[cmd_count - 1] << 8) + cmd[cmd_count - 2]);
            break;

        case 3:
            SET_MIPI_DATA((cmd[cmd_count - 1] << 16) + (cmd[cmd_count - 2] << 8) + cmd[cmd_count - 3]);
            break;

        default:
            break;
        }
```

抽象封装增加代码可读性的同时，也需要注意运行效率，比如单一寄存器的操作需要内联，参考如下：

```c
/**
* @brief  控制mipi模块按照设定的配置开始工作
* @param  none
* @retval none
*/
static inline bool drv_mipi_tx_work()
{
    MIPI_TX_PORT0->tx_control_L = 0x03;
}

/**
* @brief  获取mipi tx busy状态
* @param  none
* @retval true: MIPI TX模块处理中，false:MIPI TX模块处理完成
*/
static inline bool drv_mipi_tx_get_busy()
{
    return TAU_BIT_CHECK_ZERO(MIPI_TX_PORT0->tx_state, 0);
}

/**
* @brief  获取mipi tx read的值
* @param  none
* @retval mipi读取的值
*/
static inline uint32_t drv_mipi_tx_get_read_result()
{
    return MIPI_TX_PORT0->rd_lp_rx_data;
}

/**
* @brief  控制mipi模块写数据
* @param  data: 数据buffer
* @param  cmd_count: 数据长度
* @param  tx_mode: 发送模式 1: HS 0:LP
* @retval none
*/
void drv_mipi_tx_write_data(const uint8_t *cmd, uint8_t cmd_count, TX_MODE tx_mode);

/**
* @brief  控制mipi模块清空lp read
* @param  none
* @retval none
*/
void drv_mipi_tx_lprd_clear();
```

3. 文档存放位置

已放置蔚来芯研发部“\\weilaichip\研发部\03_研发总结和积累\00_2025 年度研发总结\李元鹏\【研发总结-李元鹏】WL818 SDK 框架规则”。