

STC8A 系列函数库说明

介绍

STC8A-SOFTWARE-LIB 开发人员工具包是一个完整的固件和软件包，包含大部分硬件接口的范例程序。本函数库基于 STC8A8K64D4 单片机开发，适用于 STC8A 系列芯片，具体的 MCU 资源，请参考用户手册中对应的章节。

目录

1. 函数库目录结构	2
1.1 应用程序模块	2
1.2 驱动程序模块	3
1.3 用户程序及配置文件	3
2. 系统流程	4
2.1 系统流程图	4
2.2 初始化程序	5
2.3 任务调度主循环	5
3. 公共宏定义	6
4. API 参考	7
4.1 STC8A_ADC	7
4.2 STC8A_Compare	8
4.3 STC8A_Delay	10
4.4 STC8A_EEPROM	10
4.5 STC8A_Exti	10
4.6 STC8A_GPIO	11
4.7 STC8A_I2C	15
4.8 STC8A_Timer	16
4.9 STC8A_UART	17
4.10 STC8A_SPI	19
4.11 STC8A_Soft_I2C	21
4.12 STC8A_Soft_UART	22
4.13 STC8A_WDT	22
4.14 STC8A_PCA	23
4.15 STC8A_PWM15bit	24
4.16 STC8A_BMM	26
4.17 STC8A_LCM	31
4.18 STC8A_NVIC	32
5. 平台配置	40

1. 函数库目录结构

```
|----App (应用程序目录)
|   |----inc (应用程序头文件目录)
|   |----src (应用程序源代码目录)
|----Driver (硬件驱动程序目录)
|   |----inc (驱动程序头文件目录)
|   |----isr (驱动中断程序目录)
|   |----src (驱动程序源代码目录)
|----RVMDK (项目及输出文件目录)
|   |----list (编译输出文件目录)
|----User (用户程序及配置文件目录)
```

1.1 应用程序模块

文件	描述
APP (.h .c)	应用程序公用变量、函数声明
APP_AD_UART (.h .c)	多路 ADC 查询采样，通过串口发送例程
APP_BMM_AD(.h .c)	STC8A8K64D4 单片机 ADC BMM 数据批量存储例程
APP_BMM_LCM	STC8A8K64D4 单片机 LCM 接口+BMM 驱动液晶屏例程
APP_BMM_M2M	STC8A8K64D4 单片机 BMM Memory to Memory 数据转移例程
APP_BMM_SPI_PS	STC8A8K64D4 单片机 UART_BMM, M2M_BMM, SPI_BMM 综合使用演示例程
APP_BMM_UART	STC8A8K64D4 单片机串口 BMM 数据批量收发存储例程
APP_INT_UART1 (.h .c)	INT0~INT4 五个唤醒源将 MCU 从休眠唤醒例程
APP_Lamp (.h .c)	使用 P6 口来演示跑马灯例程
APP_RTC (.h .c)	硬件 I2C 读写 RTC(PCF8563)例程
APP_I2C_PS (.h .c)	软件模拟 I2C 与硬件 I2C 自发自收例程
APP_SPI_PS (.h .c)	通过串口发送数据给 MCU1，MCU1 将接收到的数据由 SPI 发送给 MCU2，MCU2 再通过串口发送的透传例程
APP_WDT (.h .c)	看门狗复位使用例程
APP_EEPROM (.h .c)	STC8A8K64D4 单片机通过串口对内部自带的 EEPROM(FLASH)进行读写例程
APP_PCA.h	PCA 应用程序头文件
APP_PCA_PWM.c	PCA 输出 6, 7, 8, 10 位 PWM 呼吸灯效果例程
APP_PCA_Capture.c	PCA1 捕获 PCA0(8 位 PWM), PCA2(16 位软件定时)例程
APP_PWM15bit (.h .c)	15 位增强型 PWM 例程

1.2 驱动程序模块

文件	描述
STC8A_ADC (.h .c)	ADC 模块初始化及应用相关函数库
STC8A_Compare (.h .c)	比较器模块初始化相关函数库
STC8A_Delay (.h .c)	标准延时函数
STC8A_EEPROM (.h .c)	内部 EEPROM(Flash)模块初始化及应用相关函数库
STC8A_Exti (.h .c)	外部中断初始化相关函数库
STC8A_GPIO (.h .c)	IO 口初始化相关函数库
STC8A_I2C (.h .c)	I2C 模块初始化及应用相关函数库
STC8A_NVIC (.h .c)	嵌套向量中断控制器初始化相关函数库
STC8A_Soft_I2C (.h .c)	软件模拟 I2C 初始化及应用相关函数库
STC8A_Soft_UART (.h .c)	软件模拟 UART 初始化及应用相关函数库
STC8A_SPI (.h .c)	SPI 模块初始化及应用相关函数库
STC8A_Timer (.h .c)	定时器模块初始化及应用相关函数库
STC8A_UART (.h .c)	UART 模块初始化及应用相关函数库
STC8A_WDT (.h .c)	看门狗初始化及应用相关函数库
STC8A_PCA (.h .c)	PCA 模块初始化及应用相关函数库
STC8A_PWM15bit (.h .c)	15 位增强型 PWM 模块初始化及应用相关函数库
STC8A_BMM (.h .c)	BMM 批量数据传输模块初始化及应用相关函数库
STC8A_Switch.h	功能脚切换定义头文件
STC8A_ADC_Isr.c	ADC 模块中断函数库
STC8A_Compare_Isr.c	比较器模块中断函数库
STC8A_Exti_Isr.c	外部中断模块中断函数库
STC8A_GPIO_Isr.c	IO 口中断函数库
STC8A_I2C_Isr.c	I2C 模块中断函数库
STC8A_SPI_Isr.c	SPI 模块中断函数库
STC8A_Timer_Isr.c	定时器模块中断函数库
STC8A_UART_Isr.c	UART 模块中断函数库
STC8A_PCA_Isr.c	PCA 模块中断函数库
STC8A_PWM15bit_Isr.c	15 位增强型 PWM 模块中断函数库
STC8A_BMM_Isr.c	BMM 批量数据传输模块中断函数库

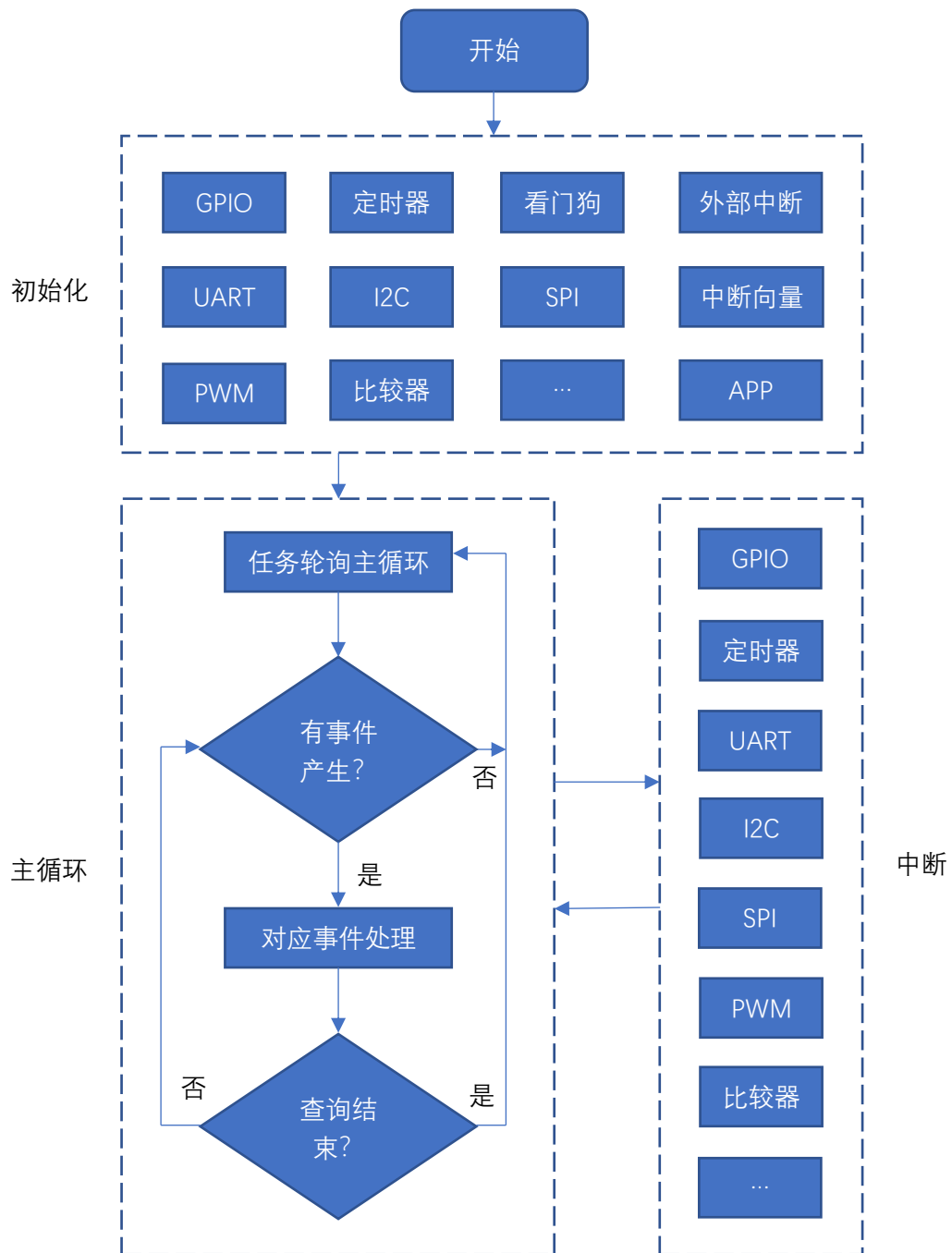
1.3 用户程序及配置文件

文件	描述
Config.h	用户配置文件, 主要是主时钟定义
Main.c	主函数文件
STC8A8K64D4.H	STC8A8K64D4 单片机寄存器定义头文件
System_init (.h .c)	系统初始化配置文件
Task (.h .c)	任务调度配置文件

Type_def.h	数据类型定义文件
isr.asm	中断号大于 31 中断，借用 13 号中断向量转换文件

2. 系统流程

2.1 系统流程图



2.2 初始化程序

“system_init.c”文件里存放系统初始化函数，执行全局初始化：

```
//=====
//                                     系统初始化
//=====
void SYS_Init(void)
{
    // GPIO_config():
    Timer_config():
    // ADC_config():
    // UART_config():
    // Exti_config():
    // I2C_config():
    // SPI_config():
    // CMP_config():
    Switch_config():
    EA = 1;

    APP_config():
}
```

2.3 任务调度主循环

系统通过定时器 0 设定 1ms 的时间作为各个任务分时调度的基准时钟。

在任务组件数组里面定义每个任务的状态、计数器、周期、执行函数：

```
static TASK_COMPONENTS Task_Comps[]=
{
    //状态 计数 周期 函数
    // {0, 250, 250, Sample_Lamp},          /* task 1 Period: 250ms */
    // {0, 200, 200, Sample_ADtoUART},     /* task 2 Period: 200ms */
    // {0, 20, 20, Sample_INTtoUART},      /* task 3 Period: 20ms */
    // {0, 1, 1, Sample_RTC},              /* task 4 Period: 1ms */
    // {0, 1, 1, Sample_I2C_PS},           /* task 5 Period: 1ms */
    // {0, 1, 1, Sample_SPI_PS},           /* task 6 Period: 1ms */
    // {0, 1, 1, Sample_EEPROM},           /* task 7 Period: 1ms */
    // {0, 100, 100, Sample_WDT},          /* task 8 Period: 100ms */
    // {0, 10, 10, Sample_PCA_PWM},        /* task 9 Period: 10ms */
    // {0, 1, 1, Sample_PCA_Capture},       /* task 10 Period: 1ms */
    // {0, 1, 1, Sample_PWM15bit},         /* task 11 Period: 1ms */
    {0, 500, 500, Sample_BMM_AD},          /* task 12 Period: 500ms */
    // {0, 500, 500, Sample_BMM_M2M},      /* task 13 Period: 100ms */
    // {0, 1, 1, Sample_BMM_UART},         /* task 14 Period: 1ms */
    // {0, 1, 1, Sample_BMM_SPI_PS},       /* task 15 Period: 1ms */
    // {0, 1, 1, Sample_BMM_LCM},         /* task 16 Period: 1ms */
    /* Add new task here */
};
```

计数器每毫秒减 1，为 0 时设置状态位，并将周期时间重载到计数器里。任务处理回调函数检查每个任务的状态，如果置位的话则执行对应的函数程序：

```

//=====
// 函数: Task_Pro_Handler_Callback
// 描述: 任务处理回调函数.
// 参数: None.
// 返回: None.
// 版本: V1.0, 2012-10-22
//=====
void Task_Pro_Handler_Callback(void)
{
    u8 i;
    for(i=0; i<Tasks_Max; i++)
    {
        if(Task_Comps[i].Run) /* If task can be run */
        {
            Task_Comps[i].Run = 0; /* Flag clear 0 */
            Task_Comps[i].TaskHook(); /* Run task */
        }
    }
}

```

执行应用范例程序，在开启任务后需要在“APP.c”文件里启动对应的初始化代码：

```

void APP_config(void)
{
    // Lamp_init();
    // ADtoUART_init();
    // INTtoUART_init();
    // RTC_init();
    // I2C_PS_init();
    // SPI_PS_init();
    // EEPROM_init();
    // WDT_init();
    // PCA_PWM_init();
    // PCA_Capture_init();
    // PWM15bit_init();
    BMM_AD_init();
    // BMM_M2M_init();
    // BMM_UART_init();
    // BMM_SPI_PS_init();
    // BMM_LCM_init();
}

```

用户可根据需要编写各种应用模块，在任务组件数组里面设定时间进行分时调度。

3. 公共宏定义

“config.h”文件进行系统时钟设置（用户可根据需要自行添加）：

```

#define MAIN_Fosc      22118400L  //定义主时钟
//#define MAIN_Fosc    12000000L  //定义主时钟
//#define MAIN_Fosc    11059200L  //定义主时钟
//#define MAIN_Fosc    5529600L   //定义主时钟
//#define MAIN_Fosc    24000000L  //定义主时钟

```

4. API 参考

4.1 STC8A_ADC

ADC 初始化函数

函数名	void ADC_Initalize(ADC_InitTypeDef *ADCx)
功能描述	ADC 初始化程序
参数	ADCx: 结构参数
返回	无

ADCx: 结构参数定义:

```
typedef struct
{
    u8  ADC_SMPduty;
    u8  ADC_Speed;
    u8  ADC_AdjResult;
    u8  ADC_CsSetup;
    u8  ADC_CsHold;
} ADC_InitTypeDef;
```

ADC_SMPduty: ADC 模拟信号采样时间控制, 设置值 0~31 (注意: SMPDUTY 一定不能设置小于 10)。

ADC_Speed: 设置 ADC 工作时钟频率

参数	功能描述
ADC_SPEED_2X1T	SYSclk/2/1
ADC_SPEED_2X2T	SYSclk/2/2
ADC_SPEED_2X3T	SYSclk/2/3
ADC_SPEED_2X4T	SYSclk/2/4
ADC_SPEED_2X5T	SYSclk/2/5
ADC_SPEED_2X6T	SYSclk/2/6
ADC_SPEED_2X7T	SYSclk/2/7
ADC_SPEED_2X8T	SYSclk/2/8
ADC_SPEED_2X9T	SYSclk/2/9
ADC_SPEED_2X10T	SYSclk/2/10
ADC_SPEED_2X11T	SYSclk/2/11
ADC_SPEED_2X12T	SYSclk/2/12
ADC_SPEED_2X13T	SYSclk/2/13
ADC_SPEED_2X14T	SYSclk/2/14
ADC_SPEED_2X15T	SYSclk/2/15
ADC_SPEED_2X16T	SYSclk/2/16

ADC_AdjResult: ADC 转换结果调整

参数	功能描述
ADC_LEFT_JUSTIFIED	转换结果左对齐
ADC_RIGHT_JUSTIFIED	转换结果右对齐

ADC_CsSetup: ADC 通道选择时间控制, 取值 0(默认), 1

ADC_CsHold: ADC 通道选择保持时间控制, 取值 0, 1(默认), 2, 3

ADC 电源控制

函数名	void ADC_PowerControl(u8 pwr)
功能描述	ADC 电源控制程序
参数	pwr: 电源控制,ENABLE 或 DISABLE.
返回	无

pwr: 电源控制

参数	功能描述
ENABLE	开启 ADC 模块电源
DISABLE	关闭 ADC 模块电源

查询法读取 ADC 转换结果

函数名	u16 Get_ADCResult(u8 channel)
功能描述	查询法读一次 ADC 结果
参数	channel: 选择要转换的 ADC 通道
返回	ADC 转换结果。返回值如果等于 4096, 表示发生错误。

channel: 设置 0~15, 分别表示 ADC0~ADC15.

4.2 STC8A_Compare

比较器初始化函数

函数名	void CMP_Inilize(CMP_InitDefine *CMPx)
功能描述	比较器初始化程序
参数	CMPx: 结构参数
返回	无

CMPx: 结构参数定义:

```
typedef struct
```

```
{
```

```
    u8  CMP_EN;
```

```
    u8  CMP_P_Select;
```

```
    u8  CMP_N_Select;
```



```

    u8 CMP_Outpt_En;
    u8 CMP_InvCMPO;
    u8 CMP_100nsFilter;
    u8 CMP_OutDelayDuty;
} CMP_InitDefine;

```

CMP_EN: 比较器使能设置

参数	功能描述
ENABLE	比较器使能
DISABLE	比较器禁止

CMP_P_Select: 比较器输入正极性选择

参数	功能描述
CMP_P_P37	选择外部端口 P3.7 做比较器正极输入源
CMP_P_P50	选择外部端口 P5.0 做比较器正极输入源
CMP_P_P51	选择外部端口 P5.1 做比较器正极输入源
CMP_P_ADC	由 ADC_CHS 所选择的 ADC 输入端做正极输入源

CMP_N_Select: 比较器输入负极性选择

参数	功能描述
CMP_N_P36	选择外部端口 P3.6 做比较器负极输入源
CMP_N_GAP	选择内部 BandGap 经过 OP 后的电压做负极输入源

CMP_Outpt_En: 比较结果输出设置

参数	功能描述
ENABLE	使能比较器结果输出，比较器结果输出到 P3.4 或者 P4.1
DISABLE	禁止比较器结果输出

CMP_InvCMPO: 比较器输出取反设置

参数	功能描述
ENABLE	使能比较器输出取反
DISABLE	禁止比较器输出取反

CMP_100nsFilter: 比较器内部 0.1uF 滤波设置

参数	功能描述
ENABLE	使能内部 0.1uF 滤波
DISABLE	禁止内部 0.1uF 滤波

CMP_OutDelayDuty: 比较结果变化延时周期数，取值 0~63。

4.3 STC8A_Delay

延时函数

函数名	void delay_ms(unsigned char ms)
功能描述	延时函数
参数	ms: 要延时的毫秒数, 这里只支持 1~255ms。自动适应主时钟。
返回	无

4.4 STC8A_EEPROM

EEPROM 读取函数

函数名	void EEPROM_read_n(u16 EE_address,u8 *DataAddress,u16 number)
功能描述	从指定 EEPROM 首地址读取若干个字节放指定的缓冲
参数 1	EE_address: 读取 EEPROM 的首地址
参数 2	DataAddress: 读取数据存放缓冲区的首地址
参数 3	number: 读取的字节长度
返回	无

EEPROM 写入函数

函数名	Void EEPROM_write_n(u16 EE_address,u8 *DataAddress,u16 number)
功能描述	把缓冲区的若干个字节写入指定首地址的 EEPROM
参数 1	EE_address: 写入 EEPROM 的首地址
参数 2	DataAddress: 写入数据源缓冲区的首地址
参数 3	number: 写入的字节长度
返回	无

EEPROM 擦除函数

函数名	void EEPROM_SectorErase(u16 EE_address)
功能描述	把指定地址的 EEPROM 扇区擦除
参数	EE_address: 要擦除的扇区 EEPROM 的地址
返回	无

4.5 STC8A_Exti

外部中断初始化函数

函数名	u8 Ext_Inilize(u8 EXT, EXT_InitTypeDef *INTx)
功能描述	外部中断初始化程序
参数 1	EXT: 外部中断号。只有 INT0, INT1 可设置中断模式, 其它默认下降沿中断。

参数 2	INTx: 结构参数
返回	成功返回 SUCCESS, 错误返回 FAIL

INTx: 结构参数定义:

```
typedef struct
{
    u8  EXTI_Mode;
} EXTI_InitTypeDef;
```

EXTI_Mode: 中断模式设置

参数	功能描述
EXT_MODE_RiseFall	上升沿+下降沿 (边沿) 中断
EXT_MODE_Fall	下降沿中断

4.6 STC8A_GPIO

IO 口初始化函数

函数名	u8 GPIO_Initalize(u8 GPIO, GPIO_InitTypeDef *GPIOx)
功能描述	初始化 IO 口
参数 1	GPIO: IO 口组号, 取值 GPIO_P0~GPIO_P7
参数 2	GPIOx: 结构参数
返回	成功返回 SUCCESS, 错误返回 FAIL

GPIOx: 结构参数定义:

```
typedef struct
{
    u8  Mode;
    u8  Pin;
} GPIO_InitTypeDef;
```

Mode: IO 模式设置

参数	功能描述
GPIO_PullUp	准双向口, 内部弱上拉, 可输入/输出, 当输入时要先写 1
GPIO_HighZ	高阻输入, 只能做输入
GPIO_OUT_OD	开漏输出, 可输入/输出, 输入/输出 1 时需要接上拉电阻
GPIO_OUT_PP	推挽输出, 只能做输出, 根据需要串接限流电阻

Pin: 要设置的端口

参数	功能描述
GPIO_Pin_0	IO 引脚 Px.0
GPIO_Pin_1	IO 引脚 Px.1
GPIO_Pin_2	IO 引脚 Px.2

GPIO_Pin_3	IO 引脚 Px.3
GPIO_Pin_4	IO 引脚 Px.4
GPIO_Pin_5	IO 引脚 Px.5
GPIO_Pin_6	IO 引脚 Px.6
GPIO_Pin_7	IO 引脚 Px.7
GPIO_Pin_LOW	Px 整组 IO 低 4 位引脚
GPIO_Pin_HIGH	Px 整组 IO 高 4 位引脚
GPIO_Pin_All	Px 整组 IO 8 位引脚

以上参数可以使用或运算，比如：

```
GPIO_InitStructure.Pin = GPIO_Pin_0 | GPIO_Pin_1 | GPIO_Pin_7;
```

宏定义方式（参数参考上表）：

1. 准双向口设置

```
P0_MODE_IO_PU (Pin); //设置 P0.x 口为准双向口
P1_MODE_IO_PU (Pin); //设置 P1.x 口为准双向口
P2_MODE_IO_PU (Pin); //设置 P2.x 口为准双向口
P3_MODE_IO_PU (Pin); //设置 P3.x 口为准双向口
P4_MODE_IO_PU (Pin); //设置 P4.x 口为准双向口
P5_MODE_IO_PU (Pin); //设置 P5.x 口为准双向口
P6_MODE_IO_PU (Pin); //设置 P6.x 口为准双向口
P7_MODE_IO_PU (Pin); //设置 P7.x 口为准双向口
```

2. 高阻输入设置

```
P0_MODE_IN_HIZ (Pin); //设置 P0.x 口为高阻输入
P1_MODE_IN_HIZ (Pin); //设置 P1.x 口为高阻输入
P2_MODE_IN_HIZ (Pin); //设置 P2.x 口为高阻输入
P3_MODE_IN_HIZ (Pin); //设置 P3.x 口为高阻输入
P4_MODE_IN_HIZ (Pin); //设置 P4.x 口为高阻输入
P5_MODE_IN_HIZ (Pin); //设置 P5.x 口为高阻输入
P6_MODE_IN_HIZ (Pin); //设置 P6.x 口为高阻输入
P7_MODE_IN_HIZ (Pin); //设置 P7.x 口为高阻输入
```

3. 开漏输出设置

```
P0_MODE_OUT_OD (Pin); //设置 P0.x 口为开漏输出
P1_MODE_OUT_OD (Pin); //设置 P1.x 口为开漏输出
P2_MODE_OUT_OD (Pin); //设置 P2.x 口为开漏输出
P3_MODE_OUT_OD (Pin); //设置 P3.x 口为开漏输出
P4_MODE_OUT_OD (Pin); //设置 P4.x 口为开漏输出
P5_MODE_OUT_OD (Pin); //设置 P5.x 口为开漏输出
P6_MODE_OUT_OD (Pin); //设置 P6.x 口为开漏输出
P7_MODE_OUT_OD (Pin); //设置 P7.x 口为开漏输出
```

4. 推挽输出设置

```
P0_MODE_OUT_PP (Pin); //设置 P0.x 口为推挽输出
P1_MODE_OUT_PP (Pin); //设置 P1.x 口为推挽输出
P2_MODE_OUT_PP (Pin); //设置 P2.x 口为推挽输出
P3_MODE_OUT_PP (Pin); //设置 P3.x 口为推挽输出
```

P4_MODE_OUT_PP (Pin); //设置 P4.x 口为推挽输出

P5_MODE_OUT_PP (Pin); //设置 P5.x 口为推挽输出

P6_MODE_OUT_PP (Pin); //设置 P6.x 口为推挽输出

P7_MODE_OUT_PP (Pin); //设置 P7.x 口为推挽输出

5. 内部 4.1K 上拉设置

P0_PULL_UP_ENABLE (Pin); //使能 P0.x 内部 4.1K 上拉

P1_PULL_UP_ENABLE (Pin); //使能 P1.x 内部 4.1K 上拉

P2_PULL_UP_ENABLE (Pin); //使能 P2.x 内部 4.1K 上拉

P3_PULL_UP_ENABLE (Pin); //使能 P3.x 内部 4.1K 上拉

P4_PULL_UP_ENABLE (Pin); //使能 P4.x 内部 4.1K 上拉

P5_PULL_UP_ENABLE (Pin); //使能 P5.x 内部 4.1K 上拉

P6_PULL_UP_ENABLE (Pin); //使能 P6.x 内部 4.1K 上拉

P7_PULL_UP_ENABLE (Pin); //使能 P7.x 内部 4.1K 上拉

P0_PULL_UP_DISABLE (Pin); //禁止 P0.x 内部 4.1K 上拉

P1_PULL_UP_DISABLE (Pin); //禁止 P1.x 内部 4.1K 上拉

P2_PULL_UP_DISABLE (Pin); //禁止 P2.x 内部 4.1K 上拉

P3_PULL_UP_DISABLE (Pin); //禁止 P3.x 内部 4.1K 上拉

P4_PULL_UP_DISABLE (Pin); //禁止 P4.x 内部 4.1K 上拉

P5_PULL_UP_DISABLE (Pin); //禁止 P5.x 内部 4.1K 上拉

P6_PULL_UP_DISABLE (Pin); //禁止 P6.x 内部 4.1K 上拉

P7_PULL_UP_DISABLE (Pin); //禁止 P7.x 内部 4.1K 上拉

6. 施密特触发设置

P0_ST_ENABLE (Pin); //使能 P0.x 施密特触发

P1_ST_ENABLE (Pin); //使能 P1.x 施密特触发

P2_ST_ENABLE (Pin); //使能 P2.x 施密特触发

P3_ST_ENABLE (Pin); //使能 P3.x 施密特触发

P4_ST_ENABLE (Pin); //使能 P4.x 施密特触发

P5_ST_ENABLE (Pin); //使能 P5.x 施密特触发

P6_ST_ENABLE (Pin); //使能 P6.x 施密特触发

P7_ST_ENABLE (Pin); //使能 P7.x 施密特触发

P0_ST_DISABLE (Pin); //禁止 P0.x 施密特触发

P1_ST_DISABLE (Pin); //禁止 P1.x 施密特触发

P2_ST_DISABLE (Pin); //禁止 P2.x 施密特触发

P3_ST_DISABLE (Pin); //禁止 P3.x 施密特触发

P4_ST_DISABLE (Pin); //禁止 P4.x 施密特触发

P5_ST_DISABLE (Pin); //禁止 P5.x 施密特触发

P6_ST_DISABLE (Pin); //禁止 P6.x 施密特触发

P7_ST_DISABLE (Pin); //禁止 P7.x 施密特触发

7. 端口电平转换速度设置

P0_SPEED_LOW (Pin); // P0.x 电平转换慢速, 相应的上下冲比较小

P1_SPEED_LOW (Pin); // P1.x 电平转换慢速, 相应的上下冲比较小

P2_SPEED_LOW (Pin); // P2.x 电平转换慢速, 相应的上下冲比较小

P3_SPEED_LOW (Pin); // P3.x 电平转换慢速, 相应的上下冲比较小
P4_SPEED_LOW (Pin); // P4.x 电平转换慢速, 相应的上下冲比较小
P5_SPEED_LOW (Pin); // P5.x 电平转换慢速, 相应的上下冲比较小
P6_SPEED_LOW (Pin); // P6.x 电平转换慢速, 相应的上下冲比较小
P7_SPEED_LOW (Pin); // P7.x 电平转换慢速, 相应的上下冲比较小

P0_SPEED_HIGH (Pin); // P0.x 电平转换快速, 相应的上下冲比较大
P1_SPEED_HIGH (Pin); // P1.x 电平转换快速, 相应的上下冲比较大
P2_SPEED_HIGH (Pin); // P2.x 电平转换快速, 相应的上下冲比较大
P3_SPEED_HIGH (Pin); // P3.x 电平转换快速, 相应的上下冲比较大
P4_SPEED_HIGH (Pin); // P4.x 电平转换快速, 相应的上下冲比较大
P5_SPEED_HIGH (Pin); // P5.x 电平转换快速, 相应的上下冲比较大
P6_SPEED_HIGH (Pin); // P6.x 电平转换快速, 相应的上下冲比较大
P7_SPEED_HIGH (Pin); // P7.x 电平转换快速, 相应的上下冲比较大

8. 端口驱动电流控制设置

P0_DRIVE_MEDIUM (Pin); // 设置 P0.x 一般驱动能力
P1_DRIVE_MEDIUM (Pin); // 设置 P1.x 一般驱动能力
P2_DRIVE_MEDIUM (Pin); // 设置 P2.x 一般驱动能力
P3_DRIVE_MEDIUM (Pin); // 设置 P3.x 一般驱动能力
P4_DRIVE_MEDIUM (Pin); // 设置 P4.x 一般驱动能力
P5_DRIVE_MEDIUM (Pin); // 设置 P5.x 一般驱动能力
P6_DRIVE_MEDIUM (Pin); // 设置 P6.x 一般驱动能力
P7_DRIVE_MEDIUM (Pin); // 设置 P7.x 一般驱动能力

P0_DRIVE_HIGH (Pin); // 设置 P0.x 增强驱动能力
P1_DRIVE_HIGH (Pin); // 设置 P1.x 增强驱动能力
P2_DRIVE_HIGH (Pin); // 设置 P2.x 增强驱动能力
P3_DRIVE_HIGH (Pin); // 设置 P3.x 增强驱动能力
P4_DRIVE_HIGH (Pin); // 设置 P4.x 增强驱动能力
P5_DRIVE_HIGH (Pin); // 设置 P5.x 增强驱动能力
P6_DRIVE_HIGH (Pin); // 设置 P6.x 增强驱动能力
P7_DRIVE_HIGH (Pin); // 设置 P7.x 增强驱动能力

9. 端口数字信号输入使能

P0_DIGIT_IN_ENABLE (Pin); // 使能 P0.x 数字信号输入
P1_DIGIT_IN_ENABLE (Pin); // 使能 P1.x 数字信号输入
P2_DIGIT_IN_ENABLE (Pin); // 使能 P2.x 数字信号输入
P3_DIGIT_IN_ENABLE (Pin); // 使能 P3.x 数字信号输入
P4_DIGIT_IN_ENABLE (Pin); // 使能 P4.x 数字信号输入
P5_DIGIT_IN_ENABLE (Pin); // 使能 P5.x 数字信号输入
P6_DIGIT_IN_ENABLE (Pin); // 使能 P6.x 数字信号输入
P7_DIGIT_IN_ENABLE (Pin); // 使能 P7.x 数字信号输入

P0_DIGIT_IN_DISABLE (Pin); // 禁止 P0.x 数字信号输入
P1_DIGIT_IN_DISABLE (Pin); // 禁止 P1.x 数字信号输入

```

P2_DIGIT_IN_DISABLE (Pin); // 禁止 P2.x 数字信号输入
P3_DIGIT_IN_DISABLE (Pin); // 禁止 P3.x 数字信号输入
P4_DIGIT_IN_DISABLE (Pin); // 禁止 P4.x 数字信号输入
P5_DIGIT_IN_DISABLE (Pin); // 禁止 P5.x 数字信号输入
P6_DIGIT_IN_DISABLE (Pin); // 禁止 P6.x 数字信号输入
P7_DIGIT_IN_DISABLE (Pin); // 禁止 P7.x 数字信号输入

```

4.7 STC8A_I2C

宏定义

```

#define I2C_BUF_LENGTH 8 //设置 I2C 数据缓冲区大小。
#define SLAW 0xA2 //设置 I2C 设备写地址
#define SLAR 0xA3 //设置 I2C 设备读地址

```

I2C 初始化函数

函数名	void I2C_Init(I2C_InitTypeDef *I2Cx)
功能描述	I2C 初始化程序
参数	I2Cx: 结构参数
返回	无

I2Cx: 结构参数定义:

```

typedef struct
{
    u8 I2C_Speed;
    u8 I2C_Enable;
    u8 I2C_Mode;
    u8 I2C_MS_WDTA;
    u8 I2C_SL_ADR;
    u8 I2C_SL_MA;
} I2C_InitTypeDef;

```

I2C_Speed: 总线速度设置，取值 0~63。总线速度=Fosc/2/(Speed*2+4)。

I2C_Enable: 功能使能

参数	功能描述
ENABLE	使能 I2C 功能
DISABLE	禁止 I2C 功能

I2C_Mode: 主从模式选择

参数	功能描述
I2C_Mode_Master	设置为主机模式
I2C_Mode_Slave	设置为从机模式

I2C_MS_WDTA: 主机自动发送设置

参数	功能描述
ENABLE	使能主机自动发送
DISABLE	禁止主机自动发送

I2C_SL_ADR: 从机设备地址, 取值 0~127。

I2C_SL_MA: 从机设备地址比较设置

参数	功能描述
ENABLE	使能从机设备地址比较
DISABLE	禁止从机设备地址比较

I2C 写入数据函数

函数名	void I2C_WriteNbyte(u8 addr, u8 *p, u8 number)
功能描述	I2C 写入若干数据程序
参数 1	addr: 指定地址
参数 2	*p: 写入数据存储位置
参数 3	number: 写入数据个数
返回	无

I2C 读取数据函数

函数名	void I2C_ReadNbyte(u8 addr, u8 *p, u8 number)
功能描述	I2C 读取若干数据程序
参数 1	addr: 指定地址
参数 2	*p: 读取数据存储位置
参数 3	number: 读取数据个数
返回	无

4.8 STC8A_Timer

定时器初始化函数

函数名	u8 Timer_Inilize(u8 TIM, TIM_InitTypeDef *TIMx)
功能描述	定时器初始化程序
参数 1	TIM: 定时器通道, 取值 Timer0, Timer1, Timer2, Timer3, Timer4
参数 2	TIMx: 结构参数
返回	成功返回 SUCCESS, 错误返回 FAIL

TIMx: 结构参数定义:

```
typedef struct
{
    u8 TIM_Mode;
```



```

u8 TIM_ClkSource;
u8 TIM_ClkOut;
u16 TIM_Value;
u8 TIM_Run;
} TIM_InitTypeDef;

```

TIM_Mode: 工作模式设置

参数	功能描述
TIM_16BitAutoReload	配置成 16 位自动重载模式
TIM_16Bit	配置成 16 位（手动重载）模式
TIM_8BitAutoReload	配置成 8 位自动重载模式
TIM_16BitAutoReloadNoMask	配置成 16 位自动重载模式，中断自动打开，不可屏蔽

TIM_ClkSource: 时钟源设置

参数	功能描述
TIM_CLOCK_1T	配置成 1T 模式
TIM_CLOCK_12T	配置成 12T 模式
TIM_CLOCK_Ext	配置成外部信号计数器模式

TIM_ClkOut: 可编程时钟输出设置

参数	功能描述
ENABLE	使能可编程时钟输出
DISABLE	禁止可编程时钟输出

TIM_Value: 装载定时/计数器初值。

TIM_Run: 是否运行设置

参数	功能描述
ENABLE	使能定时器
DISABLE	停止定时器

4.9 STC8A_UART

宏定义

```

#define UART1 1
#define UART2 2
#define UART3 3
#define UART4 4

```

启用对应的 UART 通道，如果不使用该通道的 UART，就屏蔽对应的定义，减少系统开销。

```

#define COM_TX1_Lenth 128 //设置串口 1 数据发送缓冲区大小。
#define COM_RX1_Lenth 128 //设置串口 1 数据接收缓冲区大小。

```

```

#define COM_TX2_Lenth 16 //设置串口 2 数据发送缓冲区大小。
#define COM_RX2_Lenth 16 //设置串口 2 数据接收缓冲区大小。
#define COM_TX3_Lenth 64 //设置串口 3 数据发送缓冲区大小。
#define COM_RX3_Lenth 64 //设置串口 3 数据接收缓冲区大小。
#define COM_TX4_Lenth 32 //设置串口 4 数据发送缓冲区大小。
#define COM_RX4_Lenth 32 //设置串口 4 数据接收缓冲区大小。

#define TimeOutSet1 5 //设置串口 1 数据接收超时时间。
#define TimeOutSet2 5 //设置串口 2 数据接收超时时间。
#define TimeOutSet3 5 //设置串口 3 数据接收超时时间。
#define TimeOutSet4 5 //设置串口 4 数据接收超时时间。

```

TimeOutSet 毫秒超时没收到新的数据说明一段数据接收完成。

UART 初始化函数

函数名	u8 UART_Configuration(u8 UARTx, COMx_InitDefine *COMx)
功能描述	UART 初始化程序
参数 1	UARTx: UART 设置通道, 取值 UART1, UART2, UART3, UART4
参数 2	COMx: 结构参数
返回	无

COMx: 结构参数定义:

```

typedef struct
{
    u8 UART_Mode;
    u8 UART_BRT_Use;
    u32 UART_BaudRate;
    u8 Morecommunicate;
    u8 UART_RxEnable;
    u8 BaudRateDouble;
} COMx_InitDefine;

```

UART_Mode: 模式设置

参数	功能描述
UART_ShiftRight	串口工作于同步输出方式, 仅用于 UART1
UART_8bit_BRTx	串口工作于 8 位数据, 可变波特率
UART_9bit	串口工作于 9 位数据, 固定波特率
UART_9bit_BRTx	串口工作于 9 位数据, 可变波特率

UART_BRT_Use: 波特率发生器设置

参数	功能描述
BRT_Timer1	使用 Timer1 作为波特率发生器, 适用于 UART1
BRT_Timer2	使用 Timer2 作为波特率发生器, 适用于 UART1, UART2, UART3, UART4
BRT_Timer3	使用 Timer3 作为波特率发生器, 适用于 UART3
BRT_Timer4	使用 Timer4 作为波特率发生器, 适用于 UART4

UART_BaudRate: 波特率设置，一般设为 110 ~ 115200。

Morecommunicate: 多机通讯设置

参数	功能描述
ENABLE	使能多机通讯
DISABLE	禁止多机通讯

UART_RxEnable: 允许接收设置

参数	功能描述
ENABLE	使能接收
DISABLE	禁止接收

BaudRateDouble: 波特率加倍设置(仅用于 UART1)

参数	功能描述
ENABLE	使能波特率加倍
DISABLE	禁止波特率加倍

UART 发送字节函数

函数名	void TX1_write2buff(u8 dat) void TX2_write2buff(u8 dat) void TX3_write2buff(u8 dat) void TX4_write2buff(u8 dat)
功能描述	UART 发送一个字节数据
参数	dat: 待发送数据
返回	无

UART 发送字符串函数

函数名	void PrintString1(u8 *puts) void PrintString2(u8 *puts) void PrintString3(u8 *puts) void PrintString4(u8 *puts)
功能描述	UART 发送一串数据，遇到停止符 0 结束
参数	*puts: 待发送数据缓冲区指针
返回	无

4.10 STC8A_SPI

宏定义

```
#define SPI_BUF_LENTH 128 //设置 SPI 数据缓冲区大小。
```

SPI 初始化函数

函数名	void SPI_Init(SPI_InitTypeDef *SPIx)
-----	--------------------------------------

功能描述	SPI 初始化程序
参数	SPIx: 结构参数
返回	无

COMx: 结构参数定义:

```
typedef struct
{
    u8 SPI_Enable;
    u8 SPI_SSIG;
    u8 SPI_FirstBit;
    u8 SPI_Mode;
    u8 SPI_CPOL;
    u8 SPI_CPHA;
    u8 SPI_Speed;
} SPI_InitTypeDef;
```

SPI_Enable: 功能使能设置

参数	功能描述
ENABLE	使能 SPI 功能
DISABLE	禁用 SPI 功能

SPI_SSIG: 片选位设置

参数	功能描述
ENABLE	SS 引脚确定器件是主机还是从机
DISABLE	忽略 SS 引脚功能，使用 MSTR 确定器件是主机还是从机

SPI_FirstBit: 数据发送/接收顺序设置

参数	功能描述
SPI_MSB	先发送/接收数据的高位 (MSB)
SPI_LSB	先发送/接收数据的低位 (LSB)

SPI_Mode: 主从模式设置

参数	功能描述
SPI_Mode_Master	设置为主机模式
SPI_Mode_Slave	设置为从机模式

SPI_CPOL: SPI 时钟极性设置

参数	功能描述
SPI_CPOL_Low	SCLK 空闲时为低电平
SPI_CPOL_High	SCLK 空闲时为高电平

SPI_CPHA: SPI 时钟相位设置

参数	功能描述
----	------

SPI_CPHA_1Edge	数据在 SCLK 的后时钟沿驱动，前时钟沿采样（必须 SSIG=0）
SPI_CPHA_2Edge	数据在 SCLK 的前时钟沿驱动，后时钟沿采样

SPI_Speed: SPI 时钟频率设置

参数	功能描述
SPI_Speed_4	SCLK 频率=SYSclk/4
SPI_Speed_16	SCLK 频率=SYSclk/16
SPI_Speed_64	SCLK 频率=SYSclk/64
SPI_Speed_128	SCLK 频率=SYSclk/128

SPI 模式设置

函数名	void SPI_SetMode(u8 mode)
功能描述	SPI 设置主从模式函数
参数	mode: 指定模式, 取值 SPI_Mode_Master 或 SPI_Mode_Slave
返回	无

SPI 发送一个字节数据

函数名	void SPI_WriteByte(u8 dat)
功能描述	SPI 发送一个字节数据函数
参数	dat: 要发送的数据
返回	无

4.11 STC8A_Soft_I2C

宏定义

```
#define SLAW    0x5A    //设置模拟 I2C 设备写地址
#define SLAR    0x5B    //设置模拟 I2C 设备读地址

sbit    SDA = P0^1;    //定义模拟 I2C 的 SDA 脚
sbit    SCL = P0^0;    //定义模拟 I2C 的 SCL 脚
```

软件模拟 I2C 发送一串数据

函数名	void SI2C_WriteNbyte(u8 addr, u8 *p, u8 number)
功能描述	软件模拟 I2C 发送一串数据函数
参数 1	addr: 指定地址
参数 2	*p: 发送数据存储位置
参数 3	number: 发送数据个数
返回	无

软件模拟 I2C 读取一串数据

函数名	void SI2C_ReadNbyte(u8 addr, u8 *p, u8 number)
功能描述	软件模拟 I2C 读取一串数据函数

参数 1	addr: 指定地址
参数 2	*p: 读取数据存储位置
参数 3	number: 读取数据个数
返回	无

4.12 STC8A_Soft_UART

宏定义

```
sbit P_TXD = P3^1; //定义模拟串口发送端,可以是任意 IO
```

软件模拟 UART 发送一个字节数据

函数名	void TxSend(u8 dat)
功能描述	模拟串口发送程序,可作为测试监控用。固定串口参数: 9600,8,n,1。为避免中断影响,发送时关闭总中断。
参数	dat: 待发送的字节
返回	无

软件模拟 UART 发送一串数据

函数名	void PrintString(unsigned char code *puts)
功能描述	模拟串口发送一串字符串
参数	*puts: 要发送的字符指针
返回	无

4.13 STC8A_WDT

看门狗初始化

函数名	void WDT_Initalize(WDT_InitTypeDef *WDT)
功能描述	看门狗初始化程序
参数	WDT: 结构参数
返回	无

WDT: 结构参数定义:

```
typedef struct
{
    u8 WDT_Enable;
    u8 WDT_IDLE_Mode;
    u8 WDT_PS;
} WDT_InitTypeDef;
```

WDT_Enable: 看门狗使能设置

参数	功能描述
ENABLE	使能看门狗
DISABLE	禁止看门狗

WDT_IDLE_Mode: IDLE 模式停止计数设置

参数	功能描述
WDT_IDLE_STOP	IDLE 模式停止计数
WDT_IDLE_RUN	IDLE 模式继续计数

WDT_PS: 看门狗定时器时钟分频系数

参数	功能描述
WDT_SCALE_2	系统时钟 2 分频
WDT_SCALE_4	系统时钟 4 分频
WDT_SCALE_8	系统时钟 8 分频
WDT_SCALE_16	系统时钟 16 分频
WDT_SCALE_32	系统时钟 32 分频
WDT_SCALE_64	系统时钟 64 分频
WDT_SCALE_128	系统时钟 128 分频
WDT_SCALE_256	系统时钟 256 分频

清看门狗

函数名	void WDT_Clear (void)
功能描述	看门狗喂狗程序
参数	无
返回	无

4.14 STC8A_PCA

PCA 初始化

函数名	void PCA_Init(u8 PCA_id, PCA_InitTypeDef *PCAx)
功能描述	PCA 初始化程序
参数 1	PCA_id: PCA 序号, 取值 PCA0,PCA1,PCA2,PCA_Counter
参数 2	PCAx: 结构参数
返回	无

PCAx: 结构参数定义:

```
typedef struct
{
    u8 PCA_Clock;
    u8 PCA_PWM_Wide;
    u16 PCA_Value;
    u8 PCA_RUN;
```

```
} PCA_InitTypeDef;
```

PCA_Clock: PCA 输入时钟源设置

参数	功能描述
PCA_Clock_12T	系统时钟/12
PCA_Clock_2T	系统时钟/2
PCA_Clock_Timer0_OF	定时器 0 的溢出脉冲
PCA_Clock_ECI	ECI 脚的外部输入时钟
PCA_Clock_1T	系统时钟
PCA_Clock_4T	系统时钟/4
PCA_Clock_6T	系统时钟/6
PCA_Clock_8T	系统时钟/8

PCA_PWM_Wide: PCA 模块的 PWM 位数设置

参数	功能描述
PCA_PWM_8bit	8 位 PWM
PCA_PWM_7bit	7 位 PWM
PCA_PWM_6bit	6 位 PWM
PCA_PWM_10bit	10 位 PWM

PCA_Value: PCA 模块的 PWM 重载值/比较值。

PCA_RUN: PCA 计数器使能设置

参数	功能描述
ENABLE	启动 PCA 计数
DISABLE	停止 PCA 计数

更新 PWM 值

函数名	void UpdatePcaPwm(u8 PCA_id, u16 pwm_value)
功能描述	更新 PCA 模块的 PWM 值
参数 1	PCA_id: PCA 序号, 取值 PCA0,PCA1,PCA2,PCA_Counter
参数 2	pwm_value: 这个值是输出低电平的时间
返回	无

4.15 STC8A_PWM15bit

15 位增强型 PWM 初始化

函数名	u8 PWM15_Init(PWM15_InitTypeDef *PWMx)
功能描述	15 位增强型 PWM 初始化程序
参数	PWMx: 结构参数
返回	无

PWMx: 结构参数定义:

```
typedef struct
{
    u8  PWM_Enable;
    u8  PWM_Counter;
    u8  PWM_CInt;
    u8  PWM_Clock_Sel;
    u8  PWM_Clock_PS;
    u16 PWM_Period;
} PWM15_InitTypeDef;
```

PWM_Enable: PWM 使能设置

参数	功能描述
ENABLE	使能 PWM
DISABLE	关闭 PWM

PWM_Counter: PWM 计数器使能设置

参数	功能描述
ENABLE	使能 PWM 计数器
DISABLE	关闭 PWM 计数器

PWM_CInt: PWM 计数器归零中断使能设置

参数	功能描述
ENABLE	使能 PWM 计数器归零中断
DISABLE	关闭 PWM 计数器归零中断

PWM_Clock_Sel: PWM 时钟源选择设置

参数	功能描述
PWMn_CLK_SYS	时钟源为系统时钟分频后的时钟
PWMn_CLK_TM2	时钟源为定时器 2 的溢出脉冲

PWM_Clock_PS: 系统时钟分频参数, 取值 0~15, PWM 输入时钟源频率=SYSclk/(x+1)。

PWM_Period: PWM 周期, 取值 0~0x7fff。

PWM 通道控制寄存器设置

函数名	u8 PWMChannelCtrl(u8 PWM_id, u8 pwm_eno, u8 pwm_ini, u8 pwm_eni, u8 pwm_ent2i, u8 pwm_ent1i)
功能描述	PWM 通道控制寄存器设置
参数 1	PWM_id: PWM 通道序号. 取值 0~7
参数 2	pwm_eno: pwm 输出使能, 0 设为 GPIO, 1 设为 PWM 输出
参数 3	pwm_ini: pwm 输出端的初始电平, 0 为低电平, 1 为高电平
参数 4	pwm_eni: pwm 通道中断使能控制, 0 为关闭 PWM 中断, 1 为使能 PWM 中断

参数 5	pwm_ent2i: pwm 通道第二个触发点中断使能控制, 0 为关闭 PWM 第二个触发点中断, 1 为使能 PWM 第二个触发点中断
参数 6	pwm_ent1i: pwm 通道第一个触发点中断使能控制, 0 为关闭 PWM 第一个触发点中断, 1 为使能 PWM 第一个触发点中断
返回	成功返回 SUCCESS, 错误返回 FAIL

PWM 占空比设置

函数名	u8 PWM15Duty(u8 PWM_id, u16 dutyL, u16 dutyH)
功能描述	PWM 占空比设置
参数 1	PWM_id: PWM 通道序号. 取值 0~7
参数 2	dutyL: pwm 输出低电平位置, 取值 0~0x7fff
参数 3	dutyH: pwm 输出高电平位置, 取值 0~0x7fff
返回	成功返回 SUCCESS, 错误返回 FAIL

PWM 通道控制

函数名	u8 PWMLLevelSet(u8 PWM_id, u8 pwm_hldl, u8 pwm_hldh)
功能描述	PWM 通道控制寄存器设置
参数 1	PWM_id: PWM 通道序号. 取值 0~7
参数 2	pwm_hldl: pwm 强制输出低电平控制位, 0 正常输出, 1 强制输出低电平
参数 3	pwm_hldh: pwm 强制输出高电平控制位, 0 正常输出, 1 强制输出高电平
返回	成功返回 SUCCESS, 错误返回 FAIL

4.16 STC8A_BMM

ADC BMM 初始化

函数名	void BMM_ADC_Init(BMM_ADC_InitTypeDef *BMM)
功能描述	BMM ADC 初始化程序
参数	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct
{
    u8 BMM_Enable;
    u16 BMM_Channel;
    u16 BMM_Buffer;
    u8 BMM_Times;
} BMM_ADC_InitTypeDef;
```

BMM_Enable: BMM 使能设置

参数	功能描述
----	------

ENABLE	使能 ADC BMM
DISABLE	禁止 ADC BMM

BMM_Channel: ADC 通道使能寄存器, bit15~bit0 对应 ADC15~ADC0, 置 1 使能对应通道。

BMM_Buffer: ADC 转换数据存储地址。

BMM_Times: 每个通道转换次数

参数	数值	功能描述
ADC_1_Times	0xxx	转换 1 次
ADC_2_Times	1000	转换 2 次
ADC_4_Times	1001	转换 4 次
ADC_8_Times	1010	转换 8 次
ADC_16_Times	1011	转换 16 次
ADC_32_Times	1100	转换 32 次
ADC_64_Times	1101	转换 64 次
ADC_128_Times	1110	转换 128 次
ADC_256_Times	1111	转换 256 次

M2M BMM 初始化

函数名	void BMM_M2M_Inilize(BMM_M2M_InitTypeDef *BMM)
功能描述	BMM M2M 初始化程序
参数	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct
{
    u8  BMM_Enable;
    u16 BMM_Rx_Buffer;
    u16 BMM_Tx_Buffer;
    u8  BMM_Length;
    u8  BMM_SRC_Dir;
    u8  BMM_DEST_Dir;
} BMM_M2M_InitTypeDef;
```

BMM_Enable: BMM 使能设置

参数	功能描述
ENABLE	使能 M2M BMM
DISABLE	禁止 M2M BMM

BMM_Rx_Buffer: 接收数据存储地址。

BMM_Tx_Buffer: 发送数据存储地址。

BMM_Length: BMM 传输总字节数, 设置范围(0~255), 实际传输字节数为设置值 + 1。

BMM_SRC_Dir: 数据源地址改变方向

参数	功能描述
M2M_ADDR_INC	数据读取完成后源地址自动递增
M2M_ADDR_DEC	数据读取完成后源地址自动递减

BMM_DEST_Dir: 数据目标地址改变方向

参数	功能描述
M2M_ADDR_INC	数据写入完成后目标地址自动递增
M2M_ADDR_DEC	数据写入完成后目标地址自动递减

UART BMM 初始化

函数名	void BMM_UART_Init(u8 UARTx, BMM_UART_InitTypeDef *BMM)
功能描述	BMM UART 初始化程序
参数 1	UARTx: UART 设置通道, 取值 UART1, UART2, UART3, UART4
参数 2	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct
{
    u8 BMM_TX_Enable;
    u8 BMM_TX_Length;
    u16 BMM_TX_Buffer;

    u8 BMM_RX_Enable;
    u8 BMM_RX_Length;
    u16 BMM_RX_Buffer;
} BMM_UART_InitTypeDef;
```

BMM_TX_Enable: BMM 使能设置

参数	功能描述
ENABLE	使能串口发送 BMM
DISABLE	禁止串口发送 BMM

BMM_TX_Length: BMM 发送总字节数, 设置范围(0~255), 实际传输字节数为设置值 + 1。

BMM_Tx_Buffer: 发送数据存储地址。

BMM_RX_Enable: BMM 使能设置

参数	功能描述
ENABLE	使能串口接收 BMM
DISABLE	禁止串口接收 BMM

BMM_RX_Length: BMM 接收总字节数, 设置范围(0~255), 实际传输字节数为设置值 + 1。

BMM_RX_Buffer: 接收数据存储地址。

SPI BMM 初始化

函数名	void BMM_SPI_Initalize(BMM_SPI_InitTypeDef *BMM)
功能描述	BMM SPI 初始化程序
参数	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct
{
    u8 BMM_Enable;
    u8 BMM_Tx_Enable;
    u8 BMM_Rx_Enable;
    u16 BMM_Rx_Buffer;
    u16 BMM_Tx_Buffer;
    u8 BMM_Length;
    u8 BMM_AUTO_SS;
    u8 BMM_SS_Sel;
} BMM_SPI_InitTypeDef;
```

BMM_Enable: BMM 使能设置

参数	功能描述
ENABLE	使能 SPI BMM
DISABLE	禁止 SPI BMM

BMM_Tx_Enable: BMM 发送数据使能设置

参数	功能描述
ENABLE	使能 SPI BMM 发送数据
DISABLE	禁止 SPI BMM 发送数据

BMM_Rx_Enable: BMM 接收数据使能设置

参数	功能描述
ENABLE	使能 SPI BMM 接收数据
DISABLE	禁止 SPI BMM 接收数据

BMM_Rx_Buffer: 接收数据存储地址。

BMM_Tx_Buffer: 发送数据存储地址。

BMM_Length: BMM 传输总字节数, 设置范围(0~255), 实际传输字节数为设置值 + 1。

BMM_AUTO_SS: 自动控制 SS 脚使能设置

参数	功能描述
ENABLE	SPI BMM 传输过程中, 自动拉低 SS 脚, 传输完成后恢复原始状态
DISABLE	SPI BMM 传输过程中, 不自动控制 SS 脚

BMM_SS_Sel: 自动控制 SS 脚选择

参数	功能描述
SPI_SS_P12	选择 P1.2 作为自动控制 SS 脚
SPI_SS_P22	选择 P2.2 作为自动控制 SS 脚
SPI_SS_P74	选择 P7.4 作为自动控制 SS 脚
SPI_SS_P35	选择 P3.5 作为自动控制 SS 脚

LCM BMM 初始化

函数名	void BMM_LCM_Init(BMM_LCM_InitTypeDef *BMM)
功能描述	BMM LCM 初始化程序
参数	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct  
{  
    u8 BMM_Enable;  
    u16 BMM_Rx_Buffer;  
    u16 BMM_Tx_Buffer;  
    u8 BMM_Length;  
} BMM_LCM_InitTypeDef;
```

BMM_Enable: BMM 使能设置

参数	功能描述
ENABLE	使能 LCM BMM
DISABLE	禁止 LCM BMM

BMM_Rx_Buffer: 接收数据存储地址。

BMM_Tx_Buffer: 发送数据存储地址。

BMM_Length: BMM 传输总字节数, 设置范围(0~255), 实际传输字节数为设置值 + 1。

4.17 STC8A_LCM

LCM 初始化

函数名	void LCM_Initalize(LCM_InitTypeDef *LCM)
功能描述	BMM LCM 初始化程序
参数	BMM: 结构参数
返回	无

BMMx: 结构参数定义:

```
typedef struct
```

```
{
```

```
    u8 LCM_Enable;
```

```
    u8 LCM_Mode;
```

```
    u8 LCM_Bit_Wide;
```

```
    u8 LCM_Setup_Time;
```

```
    u8 LCM_Hold_Time;
```

```
} LCM_InitTypeDef;
```

LCM_Enable: LCM 接口使能设置

参数	功能描述
ENABLE	使能 LCM 接口
DISABLE	禁止 LCM 接口

LCM_Mode: LCM 接口模式设置

参数	功能描述
MODE_I8080	LCM 接口设置为 I8080 模式
MODE_M6800	LCM 接口设置为 M6800 模式

LCM_Bit_Wide: LCM 数据宽度设置

参数	功能描述
BIT_WIDE_8	LCM 接口设置为 8 位数据宽度
BIT_WIDE_16	LCM 接口设置为 16 位数据宽度

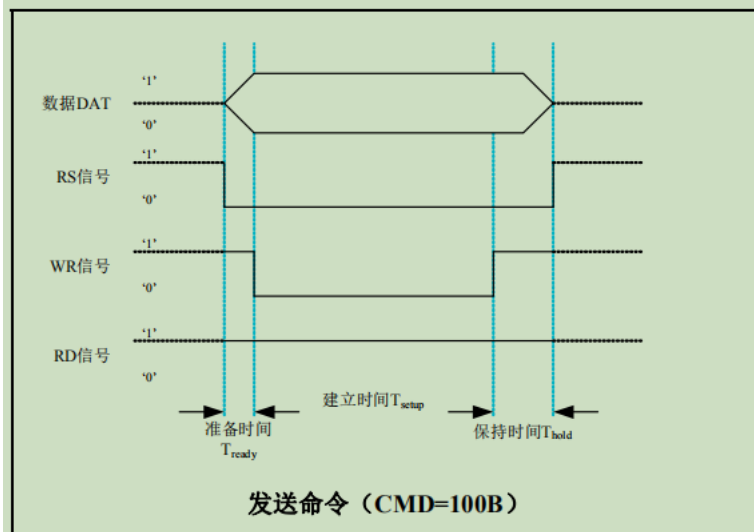
LCM_Setup_Time: LCM 通信数据建立时间，设置范围 0~7。

LCM_Hold_Time: LCM 通信数据保持时间，设置范围 0~3。

22.3 LCD接口时序图

注：
 $T_{ready} = 1$ 个系统时钟
 $T_{setup} = (SETUPT + 1)$ 个系统时钟
 $T_{hold} = (HOLDT + 1)$ 个系统时钟

22.3.1 I8080 模式



4.18 STC8A_NVIC

宏定义

```
#define FALLING_EDGE    1    //产生下降沿中断
#define RISING_EDGE    2    //产生上升沿中断
```

State: 中断使能状态

参数	功能描述
ENABLE	使能中断
DISABLE	禁止中断

Priority: 中断优先级

参数	功能描述
Polity_0	中断优先级为 0 级 (最低级)
Polity_1	中断优先级为 1 级 (较低级)
Polity_2	中断优先级为 2 级 (较高级)
Polity_3	中断优先级为 3 级 (最高级)

Timer0 嵌套向量中断

函数名	u8 NVIC_Timer0_Init(u8 State, u8 Priority)
功能描述	Timer0 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

Timer1 嵌套向量中断

函数名	u8 NVIC_Timer1_Init(u8 State, u8 Priority)
功能描述	Timer1 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

Timer2 嵌套向量中断

函数名	u8 NVIC_Timer2_Init(u8 State, u8 Priority)
功能描述	Timer2 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

Timer3 嵌套向量中断

函数名	u8 NVIC_Timer3_Init(u8 State, u8 Priority)
功能描述	Timer3 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

Timer4 嵌套向量中断

函数名	u8 NVIC_Timer4_Init(u8 State, u8 Priority)
功能描述	Timer4 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

INT0 嵌套向量中断

函数名	u8 NVIC_INT0_Init(u8 State, u8 Priority)
-----	--

功能描述	INT0 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

INT1 嵌套向量中断

函数名	u8 NVIC_INT1_Init(u8 State, u8 Priority)
功能描述	INT1 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

INT2 嵌套向量中断

函数名	u8 NVIC_INT2_Init(u8 State, u8 Priority)
功能描述	INT2 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

INT3 嵌套向量中断

函数名	u8 NVIC_INT3_Init(u8 State, u8 Priority)
功能描述	INT3 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

INT4 嵌套向量中断

函数名	u8 NVIC_INT4_Init(u8 State, u8 Priority)
功能描述	INT4 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

ADC 嵌套向量中断

函数名	u8 NVIC_ADC_Init(u8 State, u8 Priority)
功能描述	ADC 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3

返回	成功返回 SUCCESS, 错误返回 FAIL
----	-------------------------

CMP 嵌套向量中断

函数名	u8 NVIC_CMP_Init(u8 State, u8 Priority)
功能描述	比较器嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, RISING_EDGE/FALLING_EDGE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

State: 中断使能状态

参数	功能描述
DISABLE	禁止中断
RISING_EDGE	使能上升沿中断
FALLING_EDGE	使能下降沿中断

I2C 嵌套向量中断

函数名	u8 NVIC_I2C_Init(u8 State, u8 Priority)
功能描述	I2C 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, I2C_Mode_Master: ENABLE/DISABLE I2C_Mode_Slave: I2C_ESTAI/I2C_ERXI/I2C_ETXI/I2C_ESTOI/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

State: 中断使能状态

参数		功能描述
主机模式	ENABLE	使能中断
	DISABLE	禁止中断
从机模式	I2C_ESTAI	从机接收 START 信号中断
	I2C_ERXI	从机接收 1 字节数据中断
	I2C_ETXI	从机发送 1 字节数据中断
	I2C_ESTOI	从机接收 STOP 信号中断
	DISABLE	禁止中断

UART1 嵌套向量中断

函数名	u8 NVIC_UART1_Init(u8 State, u8 Priority)
功能描述	UART1 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

UART2 嵌套向量中断

函数名	u8 NVIC_UART2_Init(u8 State, u8 Priority)
功能描述	UART2 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

UART3 嵌套向量中断

函数名	u8 NVIC_UART3_Init(u8 State, u8 Priority)
功能描述	UART3 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

UART4 嵌套向量中断

函数名	u8 NVIC_UART4_Init(u8 State, u8 Priority)
功能描述	UART4 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

SPI 嵌套向量中断

函数名	u8 NVIC_SPI_Init(u8 State, u8 Priority)
功能描述	SPI 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

PCA 嵌套向量中断

函数名	u8 NVIC_PCA_Init(u8 Channel, u8 State, u8 Priority)
功能描述	PCA 嵌套向量中断控制器初始化
参数 1	Channel: 通道, PCA0/PCA1/PCA2/PCA_Counter
参数 2	State: 中断使能状态, PCA_ECOM/PCA_CCAPP/PCA_CCAPN/PCA_MAT/ PCA_TOG/PCA_PWM/PCA_ECCF/DISABLE
参数 3	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

State: 中断使能状态

参数	功能描述
DISABLE	禁止中断
PCA_ECOM	允许 PCA 模块的比较功能
PCA_CCAPP	允许 PCA 模块进行上升沿捕获
PCA_CCAPN	允许 PCA 模块进行下降沿捕获
PCA_MAT	允许 PCA 模块的匹配功能
PCA_TOG	允许 PCA 模块的高速脉冲输出功能
PCA_PWM	允许 PCA 模块的脉宽调制输出功能
PCA_ECCF	允许 PCA 模块的匹配/捕获中断

BMM ADC 嵌套向量中断

函数名	u8 NVIC_BMM_ADC_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM ADC 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM M2M 嵌套向量中断

函数名	u8 NVIC_BMM_M2M_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM M2M 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM SPI 嵌套向量中断

函数名	u8 NVIC_BMM_SPI_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM SPI 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART1 Tx 嵌套向量中断

函数名	u8 NVIC_BMM_UART1_Tx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART1 Tx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3

参数 3	Bus_Priority: 数据总线访问优先级, Priority_0,Priority_1,Priority_2,Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART1 Rx 嵌套向量中断

函数名	u8 NVIC_BMM_UART1_Rx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART1 Rx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0,Polity_1,Polity_2,Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0,Priority_1,Priority_2,Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART2 Tx 嵌套向量中断

函数名	u8 NVIC_BMM_UART2_Tx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART2 Tx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0,Polity_1,Polity_2,Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0,Priority_1,Priority_2,Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART2 Rx 嵌套向量中断

函数名	u8 NVIC_BMM_UART2_Rx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART2 Rx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0,Polity_1,Polity_2,Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0,Priority_1,Priority_2,Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART3 Tx 嵌套向量中断

函数名	u8 NVIC_BMM_UART3_Tx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART3 Tx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0,Polity_1,Polity_2,Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0,Priority_1,Priority_2,Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART3 Rx 嵌套向量中断

函数名	u8 NVIC_BMM_UART3_Rx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART3 Rx 嵌套向量中断控制器初始化

参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART4 Tx 嵌套向量中断

函数名	u8 NVIC_BMM_UART4_Tx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART4 Tx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM UART4 Rx 嵌套向量中断

函数名	u8 NVIC_BMM_UART4_Rx_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM UART4 Rx 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

BMM LCM 嵌套向量中断

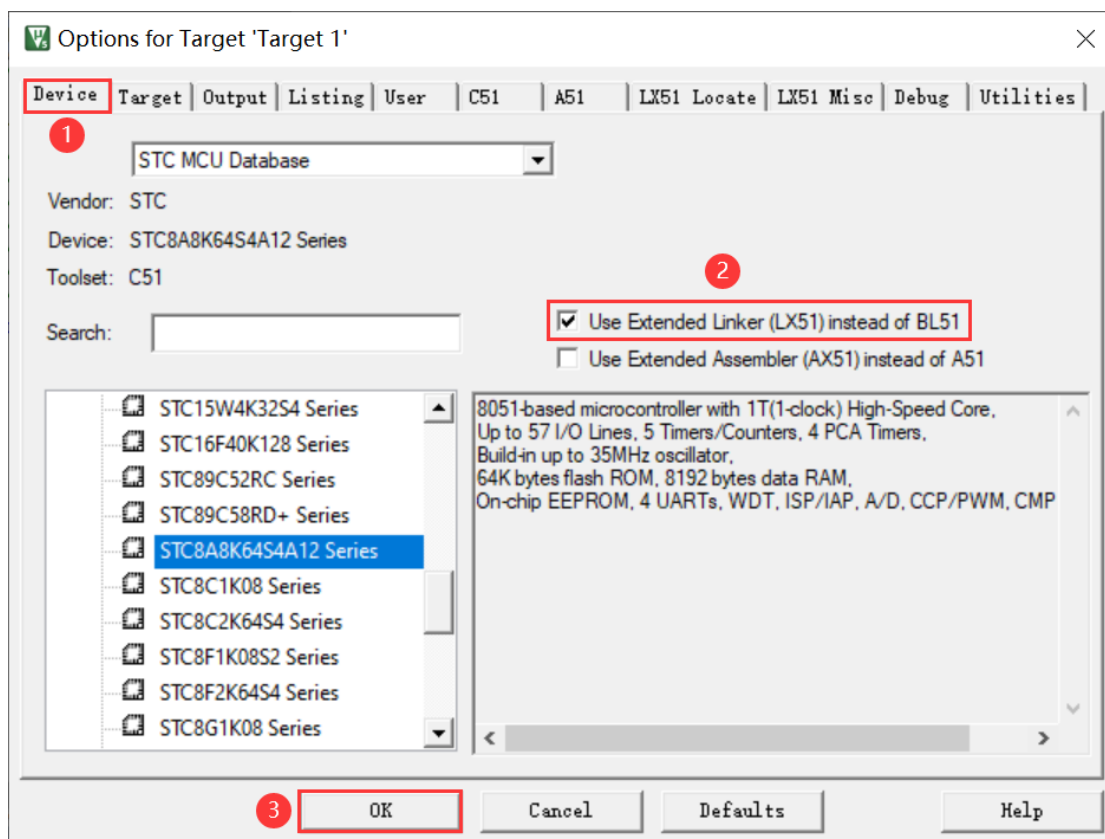
函数名	u8 NVIC_BMM_LCM_Init(u8 State, u8 Priority, u8 Bus_Priority)
功能描述	BMM LCM 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
参数 3	Bus_Priority: 数据总线访问优先级, Priority_0, Priority_1, Priority_2, Priority_3
返回	成功返回 SUCCESS, 错误返回 FAIL

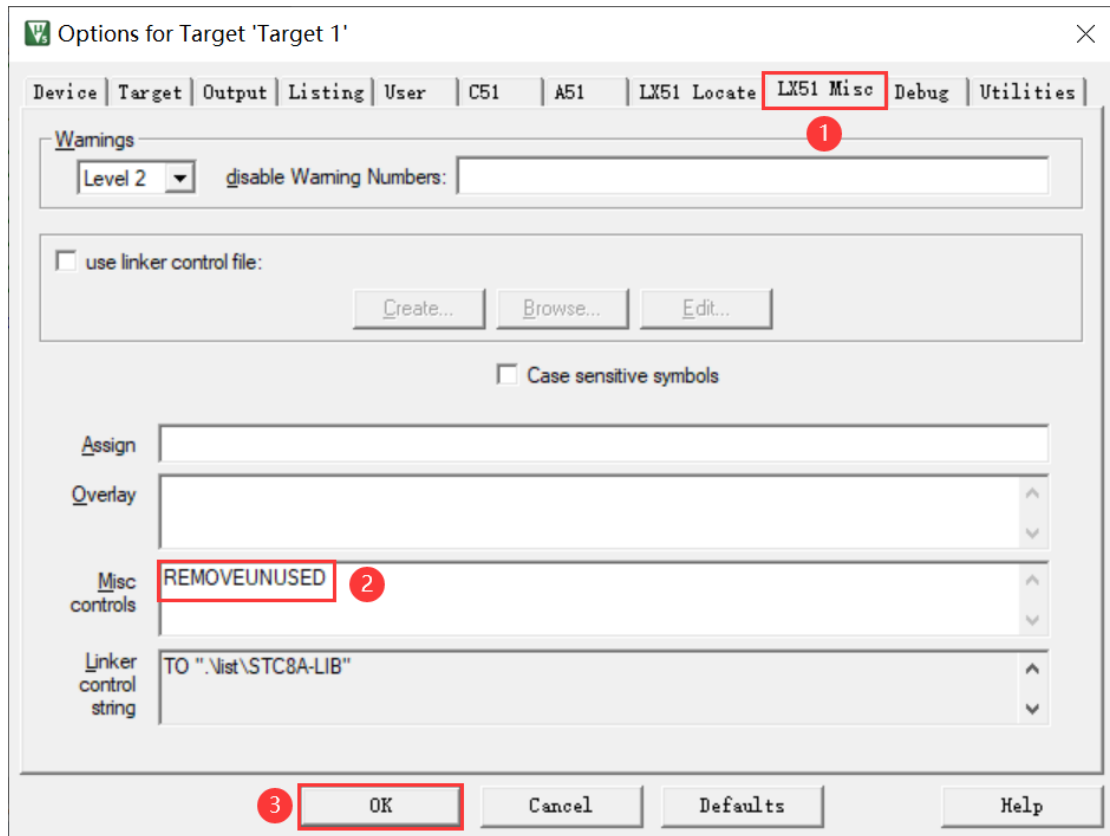
LCM 嵌套向量中断

函数名	u8 NVIC_LCM_Init(u8 State, u8 Priority)
功能描述	LCM 嵌套向量中断控制器初始化
参数 1	State: 中断使能状态, ENABLE/DISABLE
参数 2	Priority: 中断优先级, Polity_0, Polity_1, Polity_2, Polity_3
返回	成功返回 SUCCESS, 错误返回 FAIL

5. 平台配置

Keil 使用 LX51 替代 BL51，勾选后 BL51 选项卡就变成 LX51，在 LX51 Misc 选项卡 Misc controls 输入框里面添加参数 REMOVEUNUSED（不区分大小写），编译时将不会包含未调用的函数。





此外，还可以手动将一些没用到的文件设置为不参与编译，进一步降低资源消耗：

