文档编号：AN_130

上海东软载波微电子有限公司

用户手册

ES8H296 库函数

修订历史

版本	修订日期	修改概要
V1.0	2020-04-08	初版发布

地　址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com

目 录

内容目录

第1章 概述

1.1 关于本文档

本文档是 ES8H296 系列芯片固件函数库的应用笔记。函数库提供了芯片内资源与外设的驱动接口，用户使用函数库进行软件开发，可避免直接对芯片内寄存器的操作，从而缩短开发周期。本文档会对函数库中的每一个驱动接口进行描述，某些接口还会附以示例代码。

1.2 芯片简介

该产品是一款高集成度的通用 MCU 芯片，内部集成 32 位 ARM Cortex-M0 CPU 内核。内部集成多个 16 位和 32 位定时器/计数器，带红外发送调制功能的 UART 模块，兼容 7816 协议的通信接口，SPI 和 I2C 通信模块，带实时时钟模块 RTC，支持停显及闪烁功能的 LCD 驱动模块，以及用于系统电源和电池电源监测的 12 位 ADC 和 LVD 模块等外设等。

- 工作条件
  - 工作电压范围：2.5V ~ 5.5V
  - 工作温度范围：-40 ~ 85℃（工业级）
- 封装
  - LQFP64 封装（支持 56 个 I/O 端口）
- 电源
  - 系统电源输入 VDD，支持工作电压为 5V 或 3.3V 的应用系统
  - 低功耗 LVD 用于监测系统电源掉电和上电，可选择产生掉电或上电中断
  - 内部 3.0V LDO 电压输出 VR30，用于内部 ADC，需外接电容
- 时钟
  - 32.768KHz 晶体振荡器（支持外部 32.768KHz 时钟输入）作为 RTC 时钟源（内置振荡器匹配电容），可配置为系统时钟源
  - 2M~16MHz 晶体振荡器（支持外部时钟输入）可配置为系统时钟源，支持内部分频器
  - 内部 16MHz RC 振荡器可配置为系统时钟源，出厂校准精度±1%（32.768KHz 晶体振荡器工作时，支持实时调校，确保全工作范围时钟精度）
  - 内部 32KHz RC 振荡器作为 WDT 时钟源，可配置为系统时钟源
  - 支持 PLL 倍频，最大可倍频至 33MHz（用于系统时钟）。当 PLL 时钟源选择为 LOSC 或 LRC 时，倍频至约 33.55MHz（32.768KHz*1024 = 33.554432MHz）；当选择时钟源为外部或内部 4MHz 时，倍频至 32MHz（4MHz*8 = 32MHz）时钟输出。
- 内核
  - ARM Cortex-M0 32 位嵌入式处理器内核
  - 支持 SWD 串行调试接口，支持 2 个监视点（watchpoint）和 4 个断点（breakpoint）
  - 内嵌向量中断控制器 NVIC

- ◇ 支持唤醒中断控制器 WIC
- ◇ NVIC 包含一个不可屏蔽中断 NMI
- ◇ 内置 1 个 SysTick 系统定时器

◆ 硬件看门狗

- ◇ 时钟源可选择
- ◇ 支持低功耗模式下唤醒
- ◇ 超时计数溢出可选择触发中断或复位

◆ 存储器

- ◇ 128K 字节 FLASH 存储器
  - 支持 ISP 在线串行编程
  - 支持 IAP 在应用中编程
  - 支持 FLASH 编程代码加密保护
- ◇ 12K 字节 SRAM 存储器
  - SRAM 存储空间及外设寄存器地址空间支持位带（Bit band）扩展

◆ I/O 端口

- ◇ 最大支持 72 个 I/O 端口
  - PA 端口（PA0~PA31）
  - PB 端口（PB0~PB23）
- ◇ 支持 2 个大电流驱动口 PA12、PA24，最大灌电流能力 20mA
- ◇ 支持 8 路外部中断输入，触发方式可配置

◆ 定时器/计数器

- ◇ 支持 4 路 16 位定时器/计数器 T16N0，T16N1，T16N2，T16N3
- ◇ 支持 3 路 32 位定时器/计数器 T32N0，T32N1，T32N2
- ◇ 定时器/计数器带预分频器，扩展输入捕捉/输出调制 PWM 功能

◆ UART 通信接口

- ◇ 支持 4 路通信接口 UART0，UART1，UART2，UART3
- ◇ 支持全/半双工异步通信模式
- ◇ 支持传输波特率可配置
- ◇ 支持 4 字节接收和 4 字节发送缓冲器
- ◇ 支持 7 位/8 位/9 位数据格式
- ◇ 支持硬件产生奇偶校验位作为第 9 位数据传输，支持用户的第 9 位数据传输
- ◇ 支持接收帧错误标志、溢出标志、奇偶校验错误标志
- ◇ 支持数据接收和发送中断
- ◇ 支持红外发送调制输出
- ◇ 接收端口支持红外唤醒功能
- ◇ 支持接收和发送端口极性可配置

◆ EUART 通信接口

- ◇ 支持 2 路通信接口 EUART0，EUART1
- ◇ 兼容 UART 通信接口，可配置为普通 UART 模式
- ◇ 扩展支持异步半双工接收/发送（7816 模式）
- ◇ 扩展支持 8 位数据位和 1 位奇偶校验位（7816 模式）
- ◇ 扩展支持自动重发重收模式（7816 模式）
- ◇ 扩展支持可配置内部时钟输出（7816 模式）
- ◇ 扩展支持双通道通讯可配置（7816 模式）

◆ I2C 通信接口

- ◇ 支持 1 路通信接口 I2C
- ◇ 支持单主控模式/从动模式
- ◇ 支持标准的 I2C 总线协议，最大传输速率 400Kbit/s
- ◇ 支持 7 位从机地址
- ◇ 支持 4 字节接收/发送缓冲器
- ◇ 支持数据接收和发送中断
- ◇ SCL/SDA 端口支持推挽/开漏模式，开漏时必须使能内部弱上拉或使用外部上拉电阻
- ◇ SCL 端口支持时钟线自动下拉等待请求功能

◆ SPI 通信接口

- ◇ 支持 1 路通信接口 SPI
- ◇ 支持主控模式/从动模式
- ◇ 支持 4 种通信数据格式
- ◇ 支持 4 字节接收/发送缓冲器
- ◇ 支持数据接收和发送中断

◆ ADC 模数转换器

- ◇ 支持 1 路模数转换器 ADC
- ◇ 支持 12 位采样精度
- ◇ 支持 11 通道模拟输入
- ◇ 支持外部和内部参考电压选择
- ◇ 内置可调校参考电压，可从 AVREFP 复用端口输出
- ◇ 支持中断产生

◆ LCDC 液晶显示控制器

- ◇ 支持最大 8 COM x 41SEG
- ◇ 支持灰度调节功能
- ◇ 支持显示闪烁功能，闪烁频率可调

◆ RTC 实时时钟

- ◇ 仅 POR 上电复位有效，支持软件写保护，有效避免系统不稳定对时钟造成的影响

- ◇ 采用外部 32.768KHz 晶体振荡器作为时钟源
- ◇ 时钟调校可实现最大时间精度为±0.0254ppm
- ◇ 时间计数（实现小时、分钟和秒）和日历计数（实现年、月、日和星期），BCD 格式
- ◇ 提供 4 个可编程定时中断和 2 个可编程日历闹钟
- ◇ 提供一路可配置时钟输出
- ◇ 自动闰年识别，有效期到 2099 年
- ◇ 12 小时和 24 小时模式可配置
- ◇ 低功耗设计：工作电压为 3.6V 时，RTC 模块工作电流典型值为 1.5μA (最大值为 3μA)

1.3 芯片时钟树

ES8H296 具有丰富的时钟及配置系统，详见下图：

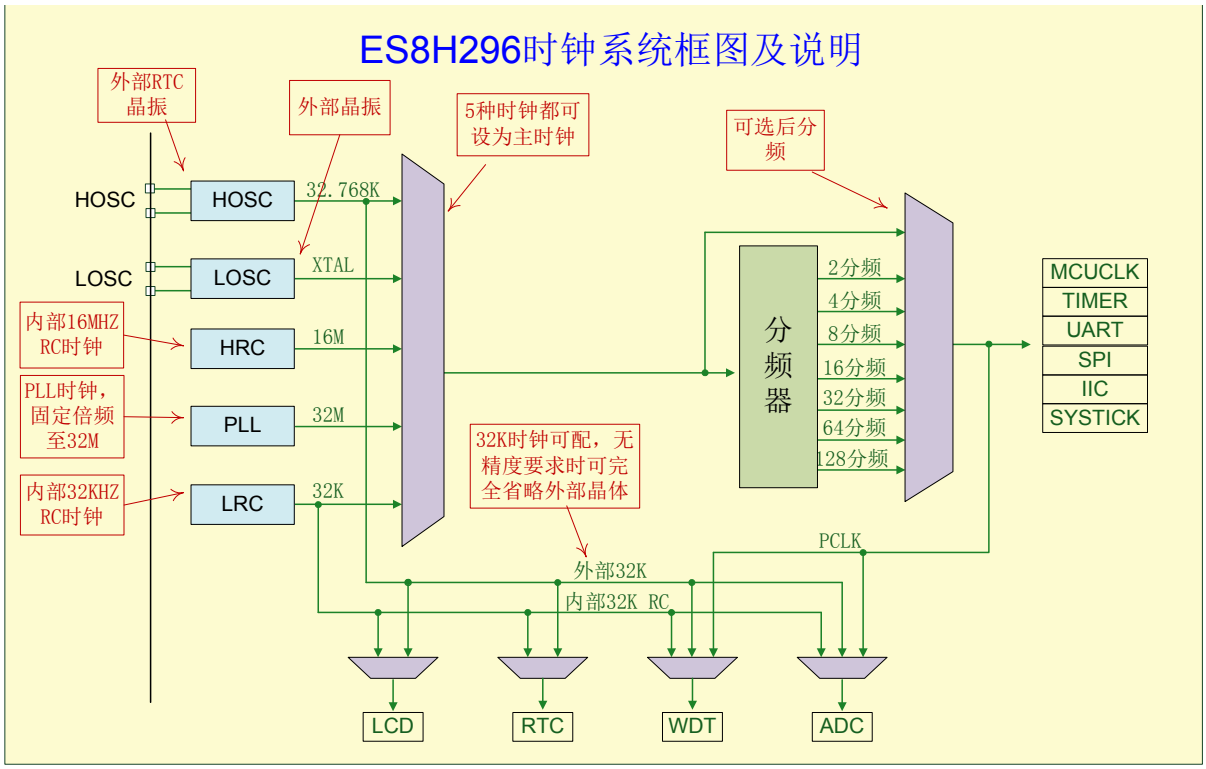

图 1-1 时钟树

1.4 文档规范

为了增强可阅读性，函数库及本文档中使用了一些缩写，函数及宏也采用规则化的命名。

1.4.1 缩写

缩写	含义	缩写	含义
Flash	闪存存储器	IIC/I2C	集成电路总线
IAP	应用中自编程	SCU	系统控制单元
ADC	模数转换器	WDT	看门狗
LCDC	液晶显示控制器	SPI	串行外设接口
UART	通用异步收发器	T16N	16 位定时器/计数器
GPIO	通用输入输出接口	T32N	32 位定时器/计数器
NVIC	嵌套中断向量列表控制器	PINT	外部端口中断
SysTick	系统滴答定时器	EUART	增强型通用异步收发器

表 1-1 缩写定义

1.4.2 命名规则

命名	功能
XXXX_Init	XXXX 外设初始化
XXXX_ITConfig	XXXX 外设中断配置
XXXX_GetFlagStatus	XXXX 外设获取标志位
XXXX_GetITStatus	XXXX 外设获取中断状态
XXXX_ClearITPendingBit	XXXX 外设清除中断标志位
XXXX_Enable	使能 XXXX
XXXX_Disable	失能 XXXX

表 1-2 命名规则

1.4.3 数据类型

函数库引用了标准 C 库中的头文件 stdint.h，其中定义了如下的数据类型：

```
typedef   signed          char int8_t;
typedef   signed short    int int16_t;
typedef   signed          int int32_t;
typedef   signed          __int64 int64_t;
typedef unsigned          char uint8_t;
typedef unsigned short    int uint16_t;
typedef unsigned          int uint32_t;
typedef unsigned          __int64 uint64_t;
typedef   signed          char int_least8_t;
typedef   signed short    int int_least16_t;
typedef   signed          int int_least32_t;
typedef   signed          __int64 int_least64_t;
typedef unsigned          char uint_least8_t;
```

```
typedef unsigned short   int uint_least16_t;
typedef unsigned         int uint_least32_t;
typedef unsigned         __int64 uint_least64_t;
typedef  signed          int int_fast8_t;
typedef  signed          int int_fast16_t;
typedef  signed          int int_fast32_t;
typedef  signed          __int64 int_fast64_t;
typedef unsigned         int uint_fast8_t;
typedef unsigned         int uint_fast16_t;
typedef unsigned         int uint_fast32_t;
typedef unsigned         __int64 uint_fast64_t;
typedef  signed          int intptr_t;
typedef unsigned         int uintptr_t;
typedef  signed          __int64 intmax_t;
typedef unsigned         __int64 uintmax_t;
```

在函数库的 system_ES8H296.h 文件中定义了几种常用的类型。

- 功能的配置类型：使能（ENABLE）或失能（DISABLE）

```
typedef  enum
{
DISABLE = 0x0,
ENABLE = 0x1
} TYPE_FUNCEN;
```

- 功能的状态类型：使能（ENABLE）或失能（DISABLE）

```
typedef  enum
{
DISABLE = 0,
ENABLE = !DISABLE
} FuncState;
```

- 标志位状态类型、中断状态类型、引脚状态类型：置位（SET）或重置（RESET）

```
typedef  enum
{
RESET = 0,
SET = !RESET
} FlagStatus, ITStatus, PinStatus;
```

- 错误状态类型：成功（SUCCES）或出错（ERROR）

```
typedef enum
{
ERROR = 0,
```

```
    SUCCESS = !ERROR
    } ErrorStatus;
```

第2章 开始使用

2.1 文件结构

函数库的文件夹结构如图 2-1 所示。

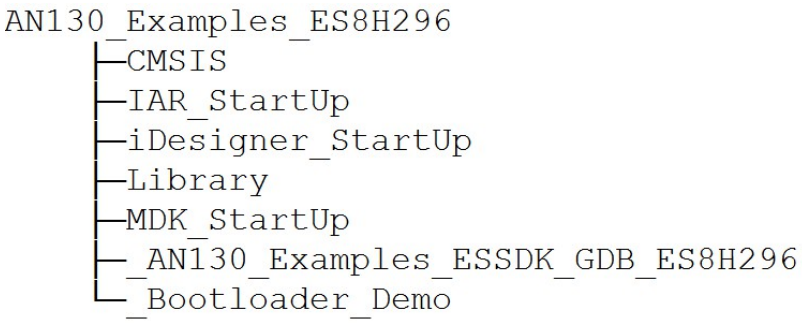

图 2-1 函数库文件夹结构

- 文件夹 CMSIS

  该文件夹下存放 ARM 内核头文件 core_cm0.h，同时也存放了芯片的头文件 ES8H296.h。

- 文件夹 Library

  该文件夹下存放函数库的源代码及头文件，下有两个子文件夹，Include 内存放头文件，Source 内存放源代码。

- 文件夹 iDesigner_StartUp/MDK_StartUp/IAR_StartUp

  该文件夹下存放芯片的启动文件 startup_ES8H296.s。

- 文件夹_AN130_Examples_ESSDK_GDB_Examples

  该文件夹下存放基于 ESSDK_GDB 开发板的演示程序，包含 ADC、按键、Flash IAP、IIC、SPI、UART、LCD、LED 等多个例程。

- 文件夹_Bootloader_Demo

  该文件夹下存放了 bootloader 相关的例程。

2.2 函数库的配置

为使函数库正常的工作，需要做一些配置。所有的配置都是在 system_ES8H296.h 文件中和 lib_config.h 文件中进行的。

2.2.1 printf函数使用串口的选择

Library\Source 目录下的 lib_printf.c 文件中重定义了微库中的函数 fputc，该函数可以将 printf 函数所需要打印的内容发送至串口，通过宏定义__PRINTF_USE_UARTx__来选择使用哪一个串口打印，例如 demo 中使用的是 UART0，则定义__PRINTF_USE_UART0__。如果不定义任何宏，则程序默认使用 UART0。

注意：UART_printf 函数采用预编译的方式，在 keil 环境下调用 UART_printf 实际上就是调用

printf 函数；在 iDesigner 下调用 UART_printf 函数即内部实现类似于 printf 的功能，但是此时的函数所提供的功能并不全面，目前只支持的转义字符及格式字符为：'\r'、'\n'、'%d'、'%s'。

2. 2. 2 函数库的引用

所有的库函数都声明于对应的外设模块头文件中，lib_config.h 文件包含了所有这些外设模块的头文件，用户在程序包含此头文件便可实现对函数库的调用，lib_config.h 中包含的头文件如下所示：

```
#include "lib_adc.h"
#include "lib_euart.h"
#include "lib_iic.h"
#include "lib_scs.h"
#include "lib_scu.h"
#include "lib_spi.h"
#include "lib_timer.h"
#include "lib_uart.h"
#include "lib_wdt.h"
#include "lib_flashiap.h"
#include "lib_gpio.h"
#include "lib_printf.h"
#include "lib_lcd.h"
```

用户可在此文件中选择所需要包含的头文件，如在未用到 ADC 外设模块时，则可在该文件中去掉对 ADC 模块头文件的包含，如下：

```
//#include "lib_adc.h"
......
```

2. 3 中断函数

中断函数的命名是固定的，错误的命名将导致无法进入中断函数。其中：NMI、HardFault、SVC、PendSV、SysTick 相关的中断函数已经分别定义和声明在 irqhandler.c 和 irqhandler.h 文件中。外设相关的中断函数按照以下表格命名：

函数名	描述
PINT0_IRQHandler	外部中断 0
PINT1_IRQHandler	外部中断 1
PINT2_IRQHandler	外部中断 2
PINT3_IRQHandler	外部中断 3
PINT4_IRQHandler	外部中断 4
PINT5_IRQHandler	外部中断 5
PINT6_IRQHandler	外部中断 6
PINT7_IRQHandler	外部中断 7
T16N0_IRQHandler	16 位定时器 0 中断

T16N1_IRQHandler	16 位定时器 1 中断
T16N2_IRQHandler	16 位定时器 2 中断
T16N3_IRQHandler	16 位定时器 3 中断
T32N0_IRQHandler	32 位定时器 0 中断
T32N1_IRQHandler	32 位定时器 1 中断
T32N2_IRQHandler	32 位定时器 2 中断
WDT_IRQHandler	看门狗模块中断
RTC_IRQHandler	RTC 模块中断
ADC_IRQHandler	ADC 模块中断
LCD_IRQHandler	LCD 控制器模块中断
LVD_IRQHandler	低电压检测模块中断
UART0_IRQHandler	串口 0 中断
UART1_IRQHandler	串口 1 中断
UART2_IRQHandler	串口 2 中断
UART3_IRQHandler	串口 3 中断
EUART0_IRQHandler	增强型串口 0 中断
SPI_IRQHandler	SPI 模块中断
IIC_IRQHandler	IIC 模块中断

表 2-1 中断函数命名

第3章 系统控制单元（SCU）

3.1 功能概述

- 支持一个不可屏蔽中断 NMI
- 支持中断向量表重映射
- 支持 LVD 低电压监测
- 内部 16MHz RC 振荡器可配置为系统时钟源
- 内部 32KHz RC 振荡器可配置为系统时钟源，用于支持低功耗系统运行
- 支持 PLL 时钟，固定倍频至 32MHz 时钟输出

3.2 特殊说明

SCU 模块的所有寄存器都受到了写保护。因此，除特别说明外，所有对 SCU 模块的操作都需要先调用"SCU_RegUnLock()"除写保护，操作完成后调用"SCU_RegLock()"来使能写保护。

3.3 寄存器结构

系统控制单元的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
 __IO SCU_PROT_Typedef PROT;
 __IO SCU_NMIC_Typedef NMIC;
 __IO SCU_PWRC_Typedef PWRC;
 uint32_t RESERVED0 ;
 __IO SCU_LVDC_Typedef LVDC;
 uint32_t RESERVED1[2] ;
 __I SCU_CFGWORD0_Typedef CFGWORD0;
 __IO SCU_FLASHW_Typedef FLASHW;
 uint32_t RESERVED2[7] ;
 __IO SCU_SCLKEN0_Typedef SCLKEN0;
 __IO SCU_SCLKEN1_Typedef SCLKEN1;
 __IO SCU_PCLKEN0_Typedef PCLKEN0;
 __IO SCU_PCLKEN1_Typedef PCLKEN1;
 __IO SCU_VRCON_Typedef VRCON;
 uint32_t RESERVED3[3] ;
 __IO SCU_TBLRMEN_Typedef TBLRMEN;
 __IO SCU_TBLOFFS_Typedef TBLOFFS;
} SCU_TypeDef;

#define APB_BASE (0x40000000UL)
```

```
#define SCU_BASE (APB_BASE + 0x00000)
#define SCU ((SCU_TypeDef *) SCU_BASE )
```

3.4 宏定义

系统控制单元的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_scu.h 中。

```
/* SCU 写保护控制 */
#define SCU_RegUnLock() (SCU->PROT.Word = 0x55AA6996)
#define SCU_RegLock()    (SCU->PROT.Word = 0x00000000)

/* NMI 使能控制 */
#define SCU_NMI_Enable()   (SCU->NMIC.NMIEN = 0x1)
#define SCU_NMI_Disable() (SCU->NMIC.NMIEN = 0x0)

/*-------LVD 模块-------*/

/* LVD 使能控制 */
#define SCU_LVD_Enable()   (SCU->LVDC.LVDEN = 0x1)
#define SCU_LVD_Disable() (SCU->LVDC.LVDEN = 0x0)

/* LVD 滤波使能控制 */
#define SCU_LVDFLT_Enable()   (SCU->LVDC.LVD_FLTEN = 0x1)
#define SCU_LVDFLT_Disable() (SCU->LVDN.LVD_FLTEN = 0x0)

/* LVD 触发电压选择 */
#define SCU_LVDVS_2V4() (SCU->LVDC.LVDVS = 0x0)
#define SCU_LVDVS_2V6() (SCU->LVDC.LVDVS = 0x1)
#define SCU_LVDVS_2V8() (SCU->LVDC.LVDVS = 0x2)
#define SCU_LVDVS_3V()   (SCU->LVDC.LVDVS = 0x3)
#define SCU_LVDVS_3V2() (SCU->LVDC.LVDVS = 0x4)
#define SCU_LVDVS_3V4() (SCU->LVDC.LVDVS = 0x5)
#define SCU_LVDVS_3V6() (SCU->LVDC.LVDVS = 0x6)
#define SCU_LVDVS_3V8() (SCU->LVDC.LVDVS = 0x7)
#define SCU_LVDVS_4V()   (SCU->LVDC.LVDVS = 0x8)
#define SCU_LVDVS_4V2() (SCU->LVDC.LVDVS = 0x9)
#define SCU_LVDVS_4V4() (SCU->LVDC.LVDVS = 0xA)
#define SCU_LVDVS_4V6() (SCU->LVDC.LVDVS = 0xB)
#define SCU_LVDVS_4V8() (SCU->LVDC.LVDVS = 0xC)
#define SCU_LVDVS_5V() (SCU->LVDC.LVDVS = 0xD)
#define SCU_LVDVS_LVDIN() (SCU->LVDC.LVDVS = 0xE)

/* LVD 中断使能控制 */
#define SCU_LVDIT_Enable()   (SCU->LVDC.LVDIE = 0x1)
```

```
#define SCU_LVDIT_Disable() (SCU->LVDC.LVDIE = 0x0)

/* LVD 中断标志位清除 */
#define SCU_LVDClearIFBit() (SCU->LVDC.LVDIF = 1)

/* LVD 中断产生模式选择 */
#define SCU_LVDIFS_Rise() (SCU->LVDC.LVDIFS = 0x0)        //LVDO 上升沿产生中断
#define SCU_LVDIFS_Fall() (SCU->LVDC.LVDIFS = 0x1)        //LVDO 下降沿产生中断
#define SCU_LVDIFS_High() (SCU->LVDC.LVDIFS = 0x2)        //LVDO 高电平产生中断
#define SCU_LVDIFS_Low()  (SCU->LVDC.LVDIFS = 0x3)        //LVDO 低电平产生中断
#define SCU_LVDIFS_Change() (SCU->LVDC.LVDIFS = 0x4)      //LVDO 电平变化产生中断

/* FLASH 访问等待时间选择 */
#define SCU_FlashWait_2Tclk()  (SCU->FLASHW.ACCT = 0x0)
#define SCU_FlashWait_3Tclk()  (SCU->FLASHW.ACCT = 0x1)
#define SCU_FlashWait_4Tclk()  (SCU->FLASHW.ACCT = 0x2)
#define SCU_FlashWait_5Tclk()  (SCU->FLASHW.ACCT = 0x3)
#define SCU_FlashWait_6Tclk()  (SCU->FLASHW.ACCT = 0x4)
#define SCU_FlashWait_7Tclk()  (SCU->FLASHW.ACCT = 0x5)
#define SCU_FlashWait_8Tclk()  (SCU->FLASHW.ACCT = 0x6)
#define SCU_FlashWait_9Tclk()  (SCU->FLASHW.ACCT = 0x7)
#define SCU_FlashWait_10Tclk() (SCU->FLASHW.ACCT = 0x8)
#define SCU_FlashWait_11Tclk() (SCU->FLASHW.ACCT = 0x9)
#define SCU_FlashWait_12Tclk() (SCU->FLASHW.ACCT = 0xA)
#define SCU_FlashWait_13Tclk() (SCU->FLASHW.ACCT = 0xB)
#define SCU_FlashWait_14Tclk() (SCU->FLASHW.ACCT = 0xC)
#define SCU_FlashWait_15Tclk() (SCU->FLASHW.ACCT = 0xD)
#define SCU_FlashWait_16Tclk() (SCU->FLASHW.ACCT = 0xE)
#define SCU_FlashWait_17Tclk() (SCU->FLASHW.ACCT = 0xF)

/*  系统时钟后分频选择 */
#define SCU_SysClk_Div1()    (SCU->SCLKEN0.SYSCLK_DIV = 0)
#define SCU_SysClk_Div2()    (SCU->SCLKEN0.SYSCLK_DIV = 1)
#define SCU_SysClk_Div4()    (SCU->SCLKEN0.SYSCLK_DIV = 2)
#define SCU_SysClk_Div8()   (SCU->SCLKEN0.SYSCLK_DIV = 3)
#define SCU_SysClk_Div16()   (SCU->SCLKEN0.SYSCLK_DIV = 4)
#define SCU_SysClk_Div32()   (SCU->SCLKEN0.SYSCLK_DIV = 5)
#define SCU_SysClk_Div64()   (SCU->SCLKEN0.SYSCLK_DIV = 6)
#define SCU_SysClk_Div128() (SCU->SCLKEN0.SYSCLK_DIV = 7)


/* HRC 使能控制 (内部 16Mhz) */
#define SCU_HRC_Enable()   (SCU->SCLKEN0.HRC_EN = 1)
#define SCU_HRC_Disable() (SCU->SCLKEN0.HRC_EN = 0)
```

```
/* HRC 调校时间间隔选择 */
#define SCU_HRC1Wait_8()   (SCU->SCLKEN0.HRC_WAIT_SEL= 0) //8 个 RTC 时钟周期
#define SCU_HRC1Wait_32() (SCU->SCLKEN0.HRC_WAIT_SEL = 1)
#define SCU_HRC1Wait_128() (SCU->SCLKEN0.HRC_WAIT_SEL = 2)
#define SCU_HRC1Wait_512() (SCU->SCLKEN0.HRC_WAIT_SEL = 3)

/* LRC 使能控制 (内部 32KHz) */
#define SCU_LRC_Enable()   (SCU->SCLKEN0.LRC_EN = 1)
#define SCU_LRC_Disable() (SCU->SCLKEN0.LRC_EN = 0)


/* HOSC 使能控制 */
#define SCU_HOSC_Enable()   (SCU->SCLKEN1.HOSC_EN = 1)
#define SCU_HOSC_Disable() (SCU->SCLKEN1.HOSC_EN = 0)

/* HOSC 振荡器稳定时间选择 */
#define SCU_HOSCTX_512()   (SCU->SCLKEN1.HOSC_TS = 0)

#define SCU_HOSCTX_1024() (SCU->SCLKEN1.HOSC_TS = 1)
#define SCU_HOSCTX_2048() (SCU->SCLKEN1.HOSC_TS = 2)

/* LOSC 低功耗模式使能控制 */
#define SCU_LOSCLP_Enable()   (SCU->SCLKEN1.LOSC_LPEN = 1)
#define SCU_LOSCLP_Disable() (SCU->SCLKEN1.LOSC_LPEN = 0)

/* PLL 模式使能控制 */
#define SCU_PLL_Enable()   (SCU->SCLKEN1.PLL_EN = 1)
#define SCU_PLL_Disable() (SCU->SCLKEN1.PLL_EN = 0)

/*  系统时钟 128 分频输出使能控制 */
#define SYSCLK_OUT_Enable()
{SCU_RegUnLock();SCU->SCLKEN0.SYSCLKOE=1;SCU_RegLock();}
#define SYSCLK_OUT_Disable()
{SCU_RegUnLock();SCU->SCLKEN0.SYSCLKOE=0;SCU_RegLock();}

/* ADC 32k 时钟选择 */
#define SCU_ADC32K_WdtClk() (SCU->SCLKEN1.ADC32K_SEL = 0)
#define SCU_ADC32K_RtcOsc() (SCU->SCLKEN1.ADC32K_SEL = 1)

/* LCD 32k 时钟选择 */
#define SCU_LCD32K_WdtClk() (SCU->SCLKEN1.LCD32K_SEL = 0)
#define SCU_LCD32K_RtcOsc() (SCU->SCLKEN1.LCD32K_SEL = 1)
```

```
/* WDT 32k 时钟选择 */
#define SCU_WDT32K_WdtClk() (SCU->SCLKEN1.WDT32K_SEL = 0)
#define SCU_WDT32K_RtcOsc() (SCU->SCLKEN1.WDT32K_SEL = 1)


/*-------外设时钟控制-------*/

/* GPIO 时钟使能控制 */
#define SCU_GPIOCLK_Enable()   (SCU->PCLKEN0.GPIO = 1)
#define SCU_GPIOCLK_Disable() (SCU->PCLKEN0.GPIO = 0)

/* FLASH IAP 时钟使能控制 */
#define SCU_IAPCLK_Enable()   (SCU->PCLKEN0.IAP = 1)
#define SCU_IAPCLK_Disable() (SCU->PCLKEN0.IAP = 0)

/* ADC 时钟使能控制 */
#define SCU_ADCCLK_Enable()   (SCU->PCLKEN0.ADC = 1)
#define SCU_ADCCLK_Disable() (SCU->PCLKEN0.ADC = 0)

/* RTC 时钟使能控制 */
#define SCU_RTCCLK_Enable()   (SCU->PCLKEN0.RTC = 1)
#define SCU_RTCCLK_Disable() (SCU->PCLKEN0.RTC = 0)

/* LCDC 时钟使能控制 */
#define SCU_LCDCCLK_Enable()   (SCU->PCLKEN0.LCDC = 1)
#define SCU_LCDCCLK_Disable() (SCU->PCLKEN0.LCDC = 0)

/* WDT 时钟使能控制 */
#define SCU_WDTCLK_Enable()   (SCU->PCLKEN0.WDT = 1)
#define SCU_WDTCLK_Disable() (SCU->PCLKEN0.WDT = 0)

/* T16N0 时钟使能控制 */
#define SCU_T16N0CLK_Enable()   (SCU->PCLKEN1.T16N0 = 1)
#define SCU_T16N0CLK_Disable() (SCU->PCLKEN1.T16N0 = 0)

/* T16N1 时钟使能控制 */
#define SCU_T16N1CLK_Enable()   (SCU->PCLKEN1.T16N1 = 1)
#define SCU_T16N1CLK_Disable() (SCU->PCLKEN1.T16N1 = 0)

/* T16N2 时钟使能控制 */
#define SCU_T16N2CLK_Enable()   (SCU->PCLKEN1.T16N2 = 1)
#define SCU_T16N2CLK_Disable() (SCU->PCLKEN1.T16N2 = 0)

/* T16N3 时钟使能控制 */
```

```
#define SCU_T16N3CLK_Enable()   (SCU->PCLKEN1.T16N3 = 1)
#define SCU_T16N3CLK_Disable() (SCU->PCLKEN1.T16N3 = 0)

/* T32N0 时钟使能控制 */
#define SCU_T32N0CLK_Enable()   (SCU->PCLKEN1.T32N0 = 1)
#define SCU_T32N0CLK_Disable() (SCU->PCLKEN1.T32N0 = 0)

/* T32N1 时钟使能控制 */
#define SCU_T32N1CLK_Enable()   (SCU->PCLKEN1.T32N1 = 1)
#define SCU_T32N1CLK_Disable() (SCU->PCLKEN1.T32N1 = 0)

/* UART0 时钟使能控制 */
#define SCU_UART0CLK_Enable()   (SCU->PCLKEN1.UART0 = 1)
#define SCU_UART0CLK_Disable() (SCU->PCLKEN1.UART0 = 0)

/* UART1 时钟使能控制 */
#define SCU_UART1CLK_Enable()   (SCU->PCLKEN1.UART1 = 1)
#define SCU_UART1CLK_Disable() (SCU->PCLKEN1.UART1 = 0)

/* UART2 时钟使能控制 */
#define SCU_UART2CLK_Enable()   (SCU->PCLKEN1.UART2 = 1)
#define SCU_UART2CLK_Disable() (SCU->PCLKEN1.UART2 = 0)

/* UART3 时钟使能控制 */
#define SCU_UART3CLK_Enable()   (SCU->PCLKEN1.UART3 = 1)
#define SCU_UART3CLK_Disable() (SCU->PCLKEN1.UART3 = 0)

/* EUART0 时钟使能控制 */
#define SCU_EUART0CLK_Enable()   (SCU->PCLKEN1.EUART0 = 1)
#define SCU_EUART0CLK_Disable() (SCU->PCLKEN1.EUART0 = 0)


/* SPI0 时钟使能控制 */
#define SCU_SPI0CLK_Enable()   (SCU->PCLKEN1.SPI = 1)
#define SCU_SPI0CLK_Disable() (SCU->PCLKEN1.SPI = 0)

/* IIC0 时钟使能控制 */
#define SCU_IIC0CLK_Enable()   (SCU->PCLKEN1.IIC = 1)
#define SCU_IIC0CLK_Disable() (SCU->PCLKEN1.IIC = 0)


/*  中断向量表重映射使能控制 */
#define SCU_TBLRemap_Enable()   (SCU->TBLRMEN.TBLREMAPEN= 1)
#define SCU_TBLRemap_Disable() (SCU->TBLRMEN.TBLREMAPEN= 0)
```

```
/* 中断向量表偏移寄存器 x 最大为 2^24=16777216 */
#define SCU_TBL_Offset(x) (SCU->TBLOFFS.TBLOFF = (uint32_t)x)
```

3. 5 库函数

系统控制块库函数定义于 lib_scu.c 中，声明于 lib_scu.h 中。

3. 5. 1 函数SCU_NMISelect

- 函数原型：void SCU_NMISelect(SCU_TYPE_NMICS NMI_Type)
- 功能描述：设置 NMI 不可屏蔽中断
- 输入参数：不可屏蔽中断类型，详见表 3-1
- 返回值：无

不可屏蔽中断枚举类型 SCU_TYPE_NMICS：

枚举元素	数值	描述
SCU_NMIIRQ_PINT0	0	外部中断 0
SCU_NMIIRQ_PINT1	1	外部中断 1
SCU_NMIIRQ_PINT2	2	外部中断 2
SCU_NMIIRQ_PINT3	3	外部中断 3
SCU_NMIIRQ_PINT4	4	外部中断 4
SCU_NMIIRQ_PINT5	5	外部中断 5
SCU_NMIIRQ_PINT6	6	外部中断 6
SCU_NMIIRQ_PINT7	7	外部中断 7
SCU_NMIIRQ_T16N0	8	定时器中断：T16N0
SCU_NMIIRQ_T16N1	9	定时器中断：T16N1
SCU_NMIIRQ_T16N2	10	定时器中断：T16N2
SCU_NMIIRQ_T16N3	11	定时器中断：T16N3
SCU_NMIIRQ_T32N0	12	定时器中断：T32N0
SCU_NMIIRQ_T32N1	13	定时器中断：T32N1
SCU_NMIIRQ_T32N2	14	定时器中断：T32N2
SCU_NMIIRQ_WDTINT	16	看门狗中断
SCU_NMIIRQ_RTCINT	17	RTC 中断
SCU_NMIIRQ_ADCINT	19	ADC 中断
SCU_NMIIRQ_LCDINT	20	LCD 控制器中断
SCU_NMIIRQ_LVD0INT	21	LVD 中断
SCU_NMIIRQ_UART0	23	串口 0 中断
SCU_NMIIRQ_UART1	24	串口 1 中断
SCU_NMIIRQ_UART2	25	串口 2 中断
SCU_NMIIRQ_UART3	26	串口 3 中断
SCU_NMIIRQ_EUART0	27	高级串口 0 中断

SCU_NMIIRQ_SPI0INT	29	SPI0 中断
SCU_NMIIRQ_IIC0INT	30	IIC0 中断

表 3-1　SCU_TYPE_NMICS

3. 5. 2　函数SCU_GetPWRCFlagStatus

- 函数原型：FlagStatus　SCU_GetPWRCFlagStatus(SCU_TYPE_PWRC　PWRC_Flag)
- 功能描述：获取 PWRC 复位状态寄存器标志位状态
- 输入参数：PWRC 寄存器标志位，详见表 3-2
- 返回值：RESET/SET

PWRC 寄存器标志位枚举类型 SCU_TYPE_PWRC：

枚举元素	数值	描述
SCU_PWRC_PORVF	0x00001	5V POR 复位标志
SCU_PWRC_PORRCF	0x00002	1.5V POR 复位标志
SCU_PWRC_PORF	0x00004	POR 总复位标志
SCU_PWRC_BORF	0x00008	BOR 复位标志
SCU_PWRC_WDTRSTF	0x00010	WDT 复位标志
SCU_PWRC_MRSTF	0x00020	MRSTn 复位标志
SCU_PWRC_SOFTRSTF	0x00040	软件复位标志
SCU_PWRC_PORLOST	0x00080	POR 丢失标志

表 3-2　SCU_TYPE_PWRC

3. 5. 3　函数SCU_ClearPWRCFlagBit

- 函数原型：void　SCU_ClearPWRCFlagBit(SCU_TYPE_PWRC　PWRC_Flag)
- 功能描述：清除 PWRC 复位状态寄存器标志位
- 输入参数：PWRC 寄存器标志位，详见表 3-2
- 返回值：无

3. 5. 4　函数SCU_GetLVDFlagStatus

- 函数原型：FlagStatus　SCU_GetLVDFlagStatus(SCU_TYPE_LVD0CON　LVD_Flag)
- 功能描述：获取 LVDD 寄存器标志位状态
- 输入参数：LVD 寄存器标志位，详见表 3-3
- 返回值：RESET/SET

LVD 寄存器标志位枚举类型 SCU_TYPE_LVD0CON：

枚举元素	数值	描述
SCU_LVDFlag_IF	0x0100	LVD 中断标志
SCU_LVDFlag_Out	0x8000	输出状态位

表 3-3　SCU_TYPE_LVD0CON

3. 5. 5　函数SCU_GetCFGWord

- 函数原型：uint32_t　SCU_GetCFGWord(void)
- 功能描述：获取芯片配置字
- 输入参数：无
- 返回值：32 位配置字

3. 5. 6　函数SCU_SysClkSelect

- 函数原型：void　SCU_SysClkSelect(SCU_TYPE_SYSCLK　Sysclk)
- 功能描述：选择系统时钟
- 输入参数：时钟源，详见表 3-4
- 返回值：无

时钟源枚举类型 SCU_TYPE_SYSCLK：

枚举元素	数值	描述
SCU_SysClk_HRC	0x0	内部 16MHZ RC 时钟
SCU_SysClk_LRC	0x1	内部 32KHZ RC 时钟
SCU_SysClk_HOSC	0x2	外部晶振时钟
SCU_SysClk_PLLCLK	0x3	PLL 锁相环倍频时钟
SCU_SysClk_LOSC	0x4	外部 RTC 时钟

表 3-4　SCU_TYPE_SYSCLK

3. 5. 7　函数SCU_GetSysClk

- 函数原型：SCU_TYPE_SYSCLK　SCU_GetSysClk(void)
- 功能描述：获取系统时钟源
- 输入参数：无
- 返回值：系统时钟源，详见表 3-4

3. 5. 8　函数SCU_SysClkChangeBusy

- 函数原型：FlagStatus　SCU_SysClkChangeBusy(void)

- 功能描述：获取系统时钟切换标志位
- 输入参数：无
- 返回值：RESET/SET

3. 5. 9 函数SCU_GetHRC1Flag

- 函数原型：FlagStatus SCU_GetHRC1Flag(void)
- 功能描述：获取 HRC1 工作标志位
- 输入参数：无
- 返回值：RESET/SET

3. 5. 10 函数SCU_GetLRCFlag

- 函数原型：FlagStatus SCU_GetLRCFlag(void)
- 功能描述：获取 LRC 工作标志位
- 输入参数：无
- 返回值：RESET/SET

3. 5. 11 函数SCU_HOSCReadyFlag

- 函数原型：FlagStatus SCU_HOSCReadyFlag(void)
- 功能描述：获取 HOSC 稳定标志位
- 输入参数：无
- 返回值：RESET（不稳定）/SET（稳定）

3. 5. 12 函数SCU_LOSCReadyFlag

- 函数原型：FlagStatus SCU_LOSCReadyFlag(void)
- 功能描述：获取 LOSC 稳定标志位
- 输入参数：无
- 返回值：RESET（不稳定）/SET（稳定）

3. 5. 13 函数SCU_PLLReadyFlag

- 函数原型：FlagStatus SCU_PLLReadyFlag(void)
- 功能描述：获取 PLL 稳定标志位

- 输入参数：无
- 返回值：RESET（不稳定）/SET（稳定）

3. 5. 14 函数SystemClockConfig

- 函数原型：void SystemClockConfig(void)
- 功能描述：系统时钟使用内部时钟 16MHZ，ADC、LCDC、WDT 32K 时钟选择内部部 32K
- 输入参数：无
- 返回值：无

注意：执行此函数前无需解除写保护，且该函数执行后不会改变写保护的状态。

3. 5. 15 函数DeviceClockAllEnable

- 函数原型：void DeviceClockAllEnable(void)
- 功能描述：打开所有外设时钟
- 输入参数：无
- 返回值：无

注意：执行此函数前无需解除写保护，且该函数执行后不会改变写保护的状态。

3. 5. 16 函数DeviceClockAllDisable

- 函数原型：void DeviceClockAllDisable(void)
- 功能描述：关闭所有外设时钟
- 输入参数：无
- 返回值：无

注意：执行此函数前无需解除写保护，且该函数执行后不会改变写保护的状态。

3. 5. 17 函数SystemClockSelect

- 函数原型：void SystemClockSelect(SCU_TYPE_SYSCLK CLKx)
- 功能描述：配置系统时钟。
- 输入参数：时钟源，详见表 3-4
- 返回值：无

注意：执行此函数前无需解除写保护，且该函数执行后不会改变写保护的状态。

3. 5. 18 函数SysclkPLL

- 函数原型：void SysclkPLL(PLL_TYPE_CLK CLKx,PLL_TYPE_UNLOCK mode)
- 功能描述：配置 PLL 时钟为系统时钟，并配置 PLL 时钟源和失锁处理。
- 输入参数：CLKx:PLL 时钟源选择，见表 3-5；mode：PLL 失锁时的处理模式，表 3-6
- 返回值：无

PLL 时钟源枚举类型 SCU_TYPE_SYSCLK：

枚举元素	数值	描述
PLL_LRC	0x0	内部 32KHZ RC 时钟
PLL_LOSC	0x1	外部 32KHz
PLL_HRC_4	0x4	内部 4MHz
PLL_HOSC_4	0x5	外接 16MHz 晶振
PLL_HOSC	0x6	外接 4MHz 晶振

表 3-5 PLL_TYPE_CLK

PLL 失锁处理枚举类型 PLL_TYPE_UNLOCK：

枚举元素	数值	描述
PLL_UNLOCK_Mode0	0x0	禁止无任何处理
PLL_UNLOCK_Mode0	0x1	PLL 时钟门控禁止，无系统时钟，等重新锁频后，提供时钟
PLL_UNLOCK_Mode0	0x2	系统时钟切换成 HRC
PLL_UNLOCK_Mode0	0x3	系统时钟切换成 HRC，PLL 锁频后自动切回

表 3-6 PLL_TYPE_UNLOCK

3. 6 函数库应用示例

```
/* 系统上电初始化 */
int main(void)
{
    SystemClockConfig();                    //配置时钟
    //DeviceClockAllEnable();               //可打开所有外设时钟
    SCU_ADCCLK_Enable();                    //打开ADC时钟
    SCU_T16N0CLK_Enable();                  //打开T16N0时钟
    SCU_UART0CLK_Enable();                  //打开UART0时钟
    UserFunction();                         //用户程序
}
```

第4章 内核模块

4.1 功能概述

内核模块包括以下三个部分：

- 系统定时器（SYSTICK）：24 位递减计数器
- 中断控制器（NVIC）
  - 支持中断嵌套
  - 支持中断向量
  - 支持中断优先级动态调整
  - 支持中断可屏蔽
- 系统控制块（SCB）：提供芯片内核系统实现的状态信息，并对内核系统工作进行控制

4.2 寄存器结构

SysTick、NVIC、SCB 的寄存器定义于文件 core_cm0.h。

```
typedef struct
{
    /* Offset: 0x000 (R/W)  SysTick Control and Status Register */
__IO uint32_t CTRL;
    /* Offset: 0x004 (R/W)  SysTick Reload Value Register   */
__IO uint32_t LOAD;
    /* Offset: 0x008 (R/W)  SysTick Current Value Register   */
__IO uint32_t VAL;
    /* Offset: 0x00C (R/ )  SysTick Calibration Register   */
__I  uint32_t CALIB;
} SysTick_Type;

typedef struct
{
    /* Offset: 0x000 (R/W)  Interrupt Set Enable Register   */
__IO uint32_t ISER[1];
uint32_t RESERVED0[31];
    /* Offset: 0x080 (R/W)  Interrupt Clear Enable Register   */
__IO uint32_t ICER[1];
uint32_t RSERVED1[31];
    /* Offset: 0x100 (R/W)  Interrupt Set Pending Register   */
__IO uint32_t ISPR[1];
uint32_t RESERVED2[31];
    /* Offset: 0x180 (R/W)  Interrupt Clear Pending Register   */
__IO uint32_t ICPR[1];
```

```
uint32_t RESERVED3[31];
uint32_t RESERVED4[64];
      /* Offset: 0x300 (R/W)   Interrupt Priority Register   */
__IO uint32_t IP[8];
}   NVIC_Type;

typedef struct
{
      /* Offset: 0x000 (R/ )   CPUID Base Register   */
__I   uint32_t CPUID;
      /* Offset: 0x004 (R/W)   Interrupt Control and State Register */
__IO uint32_t ICSR;
uint32_t RESERVED0;
      /* Offset: 0x00C (R/W) Interrupt and Reset Control Register */
__IO uint32_t AIRCR;
      /* Offset: 0x010 (R/W)   System Control Register   */
__IO uint32_t SCR;
      /* Offset: 0x014 (R/W)   Configuration Control Register   */
__IO uint32_t CCR;
uint32_t RESERVED1;
      /* Offset: 0x01C (R/W)   System Handlers Priority Registers.     */
__IO uint32_t SHP[2];
      /* Offset: 0x024 (R/W) System Handler Control Register */
__IO uint32_t SHCSR;
} SCB_Type;

#define SysTick_BASE (SCS_BASE + 0x0010UL)
#define NVIC_BASE (SCS_BASE + 0x0100UL)
#define SCB_BASE (SCS_BASE + 0x0D00UL)
#define SCB ((SCB_Type *) SCB_BASE )
#define SysTick ((SysTick_Type *) SysTick_BASE )
#define NVIC ((NVIC_Type *) NVIC_BASE )
```

4.3 宏定义

系统定时器的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_scs.h 中。

```
/* SysTick使能控制  */
#define SysTick_Enable()   (SysTick->CTRL |= 0x00000001)
#define SysTick_Disable() (SysTick->CTRL &= 0xFFFFFFFE)
```

4.4 库函数

SysTick、NVIC、SCB 库函数定义于 lib_scs.c 中，声明于 lib_scs.h 中。

4. 4. 1 函数NVIC_Init

- 函数原型：void NVIC_Init(NVIC_IRQChannel Channel,NVIC_IRQPriority Priority, TYPE_FUNCEN Cmd)
- 功能描述：NVIC 初始化配置
- 输入参数：
  - Channel：中断源选择，详见表 4-1
  - Priority：中断优先级，详见表 4-2
  - Cmd：失能或使能
- 返回值：无

中断源枚举类型 NVIC_IRQChannel：

枚举元素	数值	描述
NVIC_PINT0_IRQn	0	外部中断 0
NVIC_PINT1_IRQn	1	外部中断 1
NVIC_PINT2_IRQn	2	外部中断 2
NVIC_PINT3_IRQn	3	外部中断 3
NVIC_PINT4_IRQn	4	外部中断 4
NVIC_PINT5_IRQn	5	外部中断 5
NVIC_PINT6_IRQn	6	外部中断 6
NVIC_PINT7_IRQn	7	外部中断 7
NVIC_T16N0_IRQn	8	定时器中断：T16N0
NVIC_T16N1_IRQn	9	定时器中断：T16N1
NVIC_T16N2_IRQn	10	定时器中断：T16N2
NVIC_T16N3_IRQn	11	定时器中断：T16N3
NVIC_T32N0_IRQn	12	定时器中断：T32N0
NVIC_T32N1_IRQn	13	定时器中断：T32N1
NVIC_T32N2_IRQn	14	定时器中断：T32N2
NVIC_WDT_IRQn	16	看门狗中断
NVIC_RTC_IRQn	17	RTC 中断
NVIC_ADC_IRQn	19	ADC 中断
NVIC_LCD_IRQn	20	LCD 控制器中断
NVIC_LVD0_IRQn	21	LVD 中断
NVIC_UART0_IRQn	23	串口 0 中断
NVIC_UART1_IRQn	24	串口 1 中断
NVIC_UART2_IRQn	25	串口 2 中断
NVIC_UART3_IRQn	26	串口 3 中断
NVIC_EUART0_IRQn	27	高级串口 0 中断
NVIC_SPI0_IRQn	29	SPI0 中断
NVIC_IIC0_IRQn	30	IIC0 中断

表 4-1 NVIC_IRQChannel

中断优先级枚举类型 NVIC_IRQPriority：

枚举元素	数值	描述
NVIC_Priority_0	0	优先级 0（最高）
NVIC_Priority_1	1	优先级 1
NVIC_Priority_2	2	优先级 2
NVIC_Priority_3	3	优先级 3

表 4-2　NVIC_IRQPriority

4. 4. 2　函数SCB_SystemLPConfig

- 函数原型：void　SCB_SystemLPConfig(SCB_TYPE_SCR　LowPowerMode, TYPE_FUNCEN　NewState)
- 功能描述：配置系统休眠模式
- 输入参数：
  - ◇ LowPowerMode：休眠模式，详见表 4-3
  - ◇ NewState：使能、使能
- 返回值：无

休眠模式枚举类型 SCB_TYPE_SCR：

枚举元素	数值	描述
SCB_LP_SleepOnExit	0x02	从 ISR 中断处理程序返回到线程模式时，是否休眠
SCB_LP_SleepDeep	0x04	深度睡眠
SCB_LP_SEVOPend	0x10	中断挂起时，是否作为唤醒的选择位

表 4-3　SCB_TYPE_SCR

4. 4. 3　函数SCB_GetCpuID

- 函数原型：uint32_t　SCB_GetCpuID(void)
- 功能描述：获取 CPUID
- 输入参数：无
- 返回值：32 位 ID 值

4. 4. 4　函数SysTick_Init

- 函数原型：void　SysTick_Init(SYSTICK_InitStruType*　SysT_InitStruct)
- 功能描述：SysTick 初始化配置
- 输入参数：初始化配置结构体地址

◆ 返回值：无

初始化配置结构原型：

```
/* SysTick初始化配置结构体定义 */
typedef struct
{
    uint32_t   SysTick_Value;                  //递减数值：24位，右对齐
    SYST_TYPE_CLKS   SysTick_ClkSource;        //时钟源选择
    TYPE_FUNCEN   SysTick_ITEnable;            //中断使能、失能
 }SYSTICK_InitStruType;
```

SysTick 时钟源枚举类型 SYST_TYPE_CLKS：

枚举元素	数值	描述
SysTick_ClkS_Base	0x0	基准时钟(Hclk/3)
SysTick_ClkS_Cpu	0x1	处理器时钟(Hclk)

表 4-4　SYST_TYPE_CLKS

4.5　函数库应用示例

```
/* 初始化系统滴答定时器，定时100us中断（系统时钟为16Mhz） */
void User_SysTickInit(void)
{
    SYSTICK_InitStruType x;
    x.SysTick_ClkSource = SysTick_ClkS_Cpu;    //时钟源
    x.SysTick_Value = 1600;                    //100us
    x.SysTick_ITEnable = Enable;               //中断使能
    SysTick_Init(&x);
    NVIC_Init();
    SysTick_Enable();                          //开启定时器
}
```

第5章 通用输入输出（GPIO）

5. 1 功能概述

- 56 个双向 I/O 端口
- 支持地址扩展的 I/O 端口位操作方式
- 支持 8 路外部中断输入，触发方式可配置
- 支持 2 个大电流驱动口 PA12、PA24,最大灌电流能力 20mA。

5. 2 特殊说明

ES8H296 支持 GPIO 数量 56 个。

- PA0 ~ PA31
- PB0 ~ PB23

注意：对于不存在的引脚，请勿在程序中进行任何操作！

5. 3 寄存器结构

GPIO 的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
__IO GPIO_PADIR_Typedef PADIR;
__O GPIO_PADIRS_Typedef PADIRS;
__O GPIO_PADIRC_Typedef PADIRC;
__O GPIO_PADIRI_Typedef PADIRI;
__IO GPIO_PA_Typedef PA;
__O GPIO_PAS_Typedef PAS;
__O GPIO_PAC_Typedef PAC;
__O GPIO_PAI_Typedef PAI;
__IO GPIO_PAFUN0_Typedef PAFUN0;
__IO GPIO_PAFUN1_Typedef PAFUN1;
__IO GPIO_PAFUN2_Typedef PAFUN2;
__IO GPIO_PAFUN3_Typedef PAFUN3;
__IO GPIO_PAPUEN_Typedef PAPUEN;
__IO GPIO_PAPDEN_Typedef PAPDEN;
__IO GPIO_PAOD_Typedef PAOD;
uint32_t RESERVED0[17] ;
__IO GPIO_PBDIR_Typedef PBDIR;
__O GPIO_PBDIRS_Typedef PBDIRS;
uint32_t RESERVED1 ;
__O GPIO_PBDIRI_Typedef PBDIRI;
```

```
__IO GPIO_PB_Typedef PB;
__O GPIO_PBS_Typedef PBS;
__O GPIO_PBC_Typedef PBC;
__O GPIO_PBI_Typedef PBI;
__IO GPIO_PBFUN0_Typedef PBFUN0;
__IO GPIO_PBFUN1_Typedef PBFUN1;
__IO GPIO_PBFUN2_Typedef PBFUN2;
uint32_t RESERVED2 ;
__IO GPIO_PBPUEN_Typedef PBPUEN;
__IO GPIO_PBPDEN_Typedef PBPDEN;
__IO GPIO_PBOD_Typedef PBOD;
uint32_t RESERVED3 ;
__IO GPIO_PBSMITSEL_Typedef PBSMITSEL;
uint32_t RESERVED4[79] ;
__IO GPIO_PIE_Typedef PIE;
__IO GPIO_PIF_Typedef PIF;
__IO GPIO_PSEL_Typedef PSEL;
uint32_t RESERVED5[4] ;
uint32_t RESERVED6[54] ;
__IO GPIO_PROT_Typedef PROT;
} GPIO_TypeDef;typedef struct
#define APB_BASE (0x40000000UL)
#define GPIO_BASE (APB_BASE + 0x00400)
#define GPIO ((GPIO_TypeDef *) GPIOA_BASE )
```

5.4 宏定义

GPIO 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_gpio.h 中。

```
/**********GPIO写寄存器上锁和解锁****************/
#define GPIO_RegUnLock() (GPIO->PROT.Word = 0x78879669)        //解锁
#define GPIO_RegLock() (GPIO->PROT.Word    = 0x0)             //上锁

/* PINT使能控制  */
#define PINT0_Enable() (GPIO->PIE.PINTIE |= 0X1)
#define PINT1_Enable() (GPIO->PIE.PINTIE |= 0x2)
#define PINT2_Enable() (GPIO->PIE.PINTIE |= 0x4)
#define PINT3_Enable() (GPIO->PIE.PINTIE |= 0x8)
#define PINT4_Enable() (GPIO->PIE.PINTIE |= 0x10)
#define PINT5_Enable() (GPIO->PIE.PINTIE |= 0x20)
#define PINT6_Enable() (GPIO->PIE.PINTIE |= 0x40)
#define PINT7_Enable() (GPIO->PIE.PINTIE |= 0x80)
#define PINT0_Disable() (GPIO->PIE.PINTIE &= ~0x01)
#define PINT1_Disable() (GPIO->PIE.PINTIE &= ~0x02)
```

```
#define PINT2_Disable() (GPIO->PIE.PINTIE &= ~0x04)
#define PINT3_Disable() (GPIO->PIE.PINTIE &= ~0x08)
#define PINT4_Disable() (GPIO->PIE.PINTIE &= ~0x10)
#define PINT5_Disable() (GPIO->PIE.PINTIE &= ~0x20)
#define PINT6_Disable() (GPIO->PIE.PINTIE &= ~0x40)
#define PINT7_Disable() (GPIO->PIE.PINTIE &= ~0x80)

/* PINT屏蔽使能控制 */
#define PINT0_MaskEnable()    (GPIO->PIE.PMASK |= 0X01)
#define PINT1_MaskEnable()    (GPIO->PIE.PMASK |= 0x02)
#define PINT2_MaskEnable()    (GPIO->PIE.PMASK |= 0x04)
#define PINT3_MaskEnable()    (GPIO->PIE.PMASK |= 0x08)
#define PINT4_MaskEnable()    (GPIO->PIE.PMASK |= 0x10)
#define PINT5_MaskEnable()    (GPIO->PIE.PMASK |= 0x20)
#define PINT6_MaskEnable()    (GPIO->PIE.PMASK |= 0x40)
#define PINT7_MaskEnable()    (GPIO->PIE.PMASK |= 0x80)
#define PINT0_MaskDisable()   (GPIO->PIE.PMASK &= ~0x01)
#define PINT1_MaskDisable()   (GPIO->PIE.PMASK &= ~0x02)
#define PINT2_MaskDisable()   (GPIO->PIE.PMASK &= ~0x04)
#define PINT3_MaskDisable()   (GPIO->PIE.PMASK &= ~0x08)
#define PINT4_MaskDisable()   (GPIO->PIE.PMASK &= ~0x10)
#define PINT5_MaskDisable()   (GPIO->PIE.PMASK &= ~0x20)
#define PINT6_MaskDisable()   (GPIO->PIE.PMASK &= ~0x40)
#define PINT7_MaskDisable()   (GPIO->PIE.PMASK &= ~0x80)

/* PINT清除所有中断标记 */
#define PINT_ClearAllITPending() (GPIO->PIF.Word = (uint32_t)0xff)
```

5. 5 库函数

GPIO 库函数定义于 lib_gpio.c 中，声明于 lib_gpio.h 中。

5. 5. 1 函数GPIO_Init

- 函数原型：void GPIO_Init(GPIO_TYPE GPIOx, GPIO_TYPE_PIN PINx, GPIO_InitStruType* GPIO_InitStruct)
- 功能描述：GPIO 初始化
- 输入参数：
  - GPIOx：可以是 GPIOA/GPIOB
  - PINx：引脚选择，详见表 5-1
  - GPIO_InitStruct：初始化配置结构体地址
- 返回值：无

注意：调用此函数前请确保 GPIO 寄存器的写保护功能已经关闭。

初始化配置结构原型：

```
/* GPIO初始化配置结构体定义 */
typedef struct
{
    GPIO_TYPE_FUNC   GPIO_Func;          //引脚功能选择
    GPIO_TYPE_DIR    GPIO_Direction;     //方向选择
    TYPE_FUNCEN      GPIO_PDEN;          //上拉使能
    TYPE_FUNCEN      GPIO_PDEN;          //下拉使能
    TYPE_FUNCEN      GPIO_OD;            //输出模式开漏使能
}GPIO_InitStruType;
```

引脚选择枚举类型 GPIO_TYPE_PIN：

枚举元素	数值	描述
GPIO_Pin_0	0x00	引脚 0
GPIO_Pin_1	0x01	引脚 1
GPIO_Pin_2	0x02	引脚 2
GPIO_Pin_3	0x03	引脚 3
GPIO_Pin_4	0x04	引脚 4
GPIO_Pin_5	0x05	引脚 5
GPIO_Pin_6	0x06	引脚 6
GPIO_Pin_7	0x07	引脚 7
GPIO_Pin_8	0x08	引脚 8
GPIO_Pin_9	0x09	引脚 9
GPIO_Pin_10	0x0A	引脚 10
GPIO_Pin_11	0x0B	引脚 11
GPIO_Pin_12	0x0C	引脚 12
GPIO_Pin_13	0x0D	引脚 13
GPIO_Pin_14	0x0E	引脚 14
GPIO_Pin_15	0x0F	引脚 15
GPIO_Pin_16	0x10	引脚 16
GPIO_Pin_17	0x11	引脚 17
GPIO_Pin_18	0x12	引脚 18
GPIO_Pin_19	0x13	引脚 19
GPIO_Pin_20	0x14	引脚 20
GPIO_Pin_21	0x15	引脚 21
GPIO_Pin_22	0x16	引脚 22
GPIO_Pin_23	0x17	引脚 23
GPIO_Pin_24	0x18	引脚 24
GPIO_Pin_25	0x19	引脚 25

枚举元素	数值	描述
GPIO_Pin_26	0x1A	引脚 26
GPIO_Pin_27	0x1B	引脚 27
GPIO_Pin_28	0x1C	引脚 28
GPIO_Pin_29	0x1D	引脚 29
GPIO_Pin_30	0x1E	引脚 30
GPIO_Pin_31	0X1F	引脚 31

表 5-1　GPIO_TYPE_PIN

引脚功能选择枚举类型 GPIO_TYPE_FUNC：

枚举元素	数值	描述
GPIO_Func_0	0x0	功能 0
GPIO_Func_1	0x1	功能 1
GPIO_Func_2	0x2	功能 2
GPIO_Func_3	0x3	功能 3

表 5-2　GPIO_TYPE_FUNC

方向选择枚举类型 GPIO_TYPE_DIR：

枚举元素	数值	描述
GPIO_Dir_Out	0x0	输出
GPIO_Dir_In	0x1	输入

表 5-3　GPIO_TYPE_DIR

5. 5. 2　函数GPIO_Write

- 函数原型：GPIO_Write(GPIO_TYPE GPIOx, uint32_t Value)
- 功能描述：GPIO 端口写数据
- 输入参数：
  - GPIOx：可以是 GPIOA/GPIOB
  - Value：要写的数据（有些不存在的引脚，设置的值相对应的位是无作用的）
- 返回值：无

5. 5. 3　函数GPIO_Read

- 函数原型：uint32_t GPIO_Read(GPIO_TYPE GPIOx)
- 功能描述：GPIO 端口读数据
- 输入参数：可以是 GPIOA/GPIOB
- 返回值：读出的数据（有些不存在的引脚，读出的值相对应的位是无效的）

5. 5. 4 函数GPIO_ReadBit

- 函数原型：PinStatus GPIO_ReadBit(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx)
- 功能描述：GPIO 端口读某位数据
- 输入参数：
  - GPIOx：可以是 GPIOA/GPIOB
  - PINx：引脚选择，详见表 5-1
- 返回值：读出的数据（有些不存在的引脚，读出的值是无效的）

5. 5. 5 函数GPIOA_SetBit

- 函数原型：void GPIOA_SetBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOA 某引脚值 1
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 6 函数GPIOA_ResetBit

- 函数原型：void GPIOA_ResetBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOA 某引脚清 0
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 7 函数GPIOA_ToggleBit

- 函数原型：void GPIOA_ToggleBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOA 某引脚输出状态取反（需先设为输出模式）
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 8 函数GPIOB_SetBit

- 函数原型：void GPIOB_SetBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOB 某引脚值 1
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 9 函数GPIOB_ResetBit

- 函数原型：void GPIOB_ResetBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOB 某引脚清 0
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 10 函数GPIOB_ToggleBit

- 函数原型：void GPIOB_ToggleBit(GPIO_TYPE_PIN PINx)
- 功能描述：GPIOB 某引脚输出状态取反（需先设为输出模式）
- 输入参数：引脚选择，详见表 5-1
- 返回值：无

5. 5. 11 函数GPIOA_SetDirection

- 函数原型：void GPIOA_SetDirection(GPIO_TYPE_PIN PINx,
  GPIO_TYPE_DIR Dir_Type)
- 功能描述：GPIOA 引脚设置方向
- 输入参数：
  - PINx：引脚选择，详见表 5-1
  - Dir_Type：引脚方向选择，详见表 5-3
- 返回值：无

5. 5. 12 函数GPIOB_SetDirection

- 函数原型：void GPIOB_SetDirection(GPIO_TYPE_PIN PINx,
  GPIO_TYPE_DIR Dir_Type)
- 功能描述：GPIOA 设置方向
- 输入参数：
  - PINx：引脚选择，详见表 5-1
  - Dir_Type：引脚方向选择，详见表 5-3
- 返回值：无

5. 5. 13 函数PINT_Config

- 函数原型：void PINT_Config(PINT_TYPE PINTx, PINT_TYPE_SEL SELx,
  PINT_TYPE_TRIG TRIGx)

- 功能描述：PINT 外部中断配置
- 输入参数：
  - PINTx：外部端口中断类型，详见表 5-4
  - SELx：外部中断输入选择，详见表 5-5
  - TRIGx：外部中断触发电平选择，详见表 5-6
- 返回值：无

PINT 枚举类型 PINT_TYPE：

枚举元素	数值	描述
PINT0	0x0	外部端口中断 0
PINT1	0x1	外部端口中断 1
PINT2	0x2	外部端口中断 2
PINT3	0x3	外部端口中断 3
PINT4	0x4	外部端口中断 4
PINT5	0x5	外部端口中断 5
PINT6	0x6	外部端口中断 6
PINT7	0x7	外部端口中断 7

表 5-4　PINT_TYPE

外部中断选择枚举类型 PINT_TYPE_SEL：

枚举元素	数值	描述
PINT_SEL0	0x0	外部中断 0
PINT_SEL1	0x1	外部中断 1
PINT_SEL2	0x2	外部中断 2
PINT_SEL3	0x3	外部中断 3
PINT_SEL4	0x4	外部中断 4
PINT_SEL5	0x5	外部中断 5
PINT_SEL6	0x6	外部中断 6
PINT_SEL7	0x7	外部中断 7

表 5-5　PINT_TYPE_SEL

外部中断触发电平选择枚举类型 PINT_TYPE_TRIG：

枚举元素	数值	描述
PINT_Trig_Rise	0x0	上升沿触发
PINT_Trig_Fall	0x1	下降沿触发
PINT_Trig_High	0x2	高电平触发
PINT_Trig_Low	0x3	低电平触发
PINT_Trig_Change	0x4	电平变化触发

表 5-6　PINT_TYPE_TRIG

5. 5. 14 函数PINT_GetITStatus

- 函数原型：FlagStatus PINT_GetITStatus(PINT_TYPE_IT PINT_Flag)
- 功能描述：PINT 读取中断标志
- 输入参数：PINT 中断标志类型，详见表 5-7
- 返回值：SET/RESET

PINT 中断标志枚举类型 PINT_TYPE_IT：

枚举元素	数值	描述
PINT_IT_PINT0	0x01	PINT0
PINT_IT_PINT1	0x02	PINT1
PINT_IT_PINT2	0x04	PINT2
PINT_IT_PINT3	0x08	PINT3
PINT_IT_PINT4	0x10	PINT4
PINT_IT_PINT5	0x20	PINT5
PINT_IT_PINT6	0x40	PINT6
PINT_IT_PINT7	0x80	PINT7
PINT_IT_PINTAll	0xFF	所有 PINT

表 5-7 PINT_TYPE_IT

5. 5. 15 函数PINT_ClearITPendingBit

- 函数原型：void PINT_ClearITPendingBit(PINT_TYPE_IT PINT_Flag)
- 功能描述：PINT 清除中断标志
- 输入参数：PINT 中断标志类型，详见表 5-7
- 返回值：无

5. 6 函数库应用示例

```
int main(void)
{
    GPIO_InitStruType x;                          //定义结构体
    SystemClockConfig();                          //配置时钟
    SCU_GPIOCLK_Enable();                         //使能GPIO时钟
    GPIO_RegUnLock();                             //解除写保护
    x.GPIO_Direction = GPIO_Dir_Out;              //端口方向
    x.GPIO_Func = GPIO_Func_0;                    //端口功能
    x.GPIO_OD = DISABLE;                          //开漏
    GPIO_Init(GPIOA,GPIO_Pin_8,&x);               //初始化PA8
    GPIOA_SetBit(GPIO_Pin_8);                     //PA8高
```

```
    UserFunction1();                        //用户程序
    GPIOA_ResetBit(GPIO_Pin_8);             //PA8低
    UserFunction2();                        //用户程序
}
```

第6章 定时器/计数器（T16N/T32N）

6. 1 功能概述

ES8H296 芯片支持 4 个 16 位定时器 T16N0/1/2/3 及 3 个 32 位定时器 T32N0/1/2。

6. 1. 1 T16N

- 支持定时/计数工作模式可配置
- 支持 1 组 16 位可配置定时/计数寄存器 T16N_CNT
- 支持 1 组 8 位可配置预分频计数匹配寄存器 T16N_PREMAT
- 支持 4 组 16 位计数匹配寄存器 T16N_MAT0/T16N_MAT1/T16N_MAT2/ T16N_MAT3，计数匹配后支持下列操作：
  - ◇ 产生中断
  - ◇ 支持 T16N_CNT 计数寄存器三种操作：保持，清 0，或继续计数
  - ◇ 支持 T16N0OUT0/T16N0OUT1 端口四种操作：保持，清 0，置 1，或取反
- 支持输入捕捉功能
  - ◇ 支持捕捉边沿可配置
  - ◇ 支持捕捉次数可配置
- 支持输出调制功能 PWM

6. 1. 2 T32N

- 支持定时/计数工作模式可配置
- 支持 1 组 32 位可配置定时/计数寄存器 T32N_CNT
- 支持 1 组 8 位可配置预分频计数匹配寄存器 T32N_PREMAT
- 支持 4 组 32 位计数匹配寄存器 T32N_MAT0/T32N_MAT1/T32N_MAT2/ T32N_MAT3，计数匹配后支持下列操作：
  - ◇ 产生中断
  - ◇ 支持 T32N_CNT 计数寄存器三种操作：保持，清 0，或继续计数
  - ◇ 支持 T32N0OUT0/T32N0OUT1 端口四种操作：保持，清 0，置 1，或取反
- 支持输入捕捉功能
  - ◇ 支持捕捉边沿可配置
  - ◇ 支持捕捉次数可配置
- 支持输出调制功能 PWM

6. 2 寄存器结构

定时器/计数器模块的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
 __IO T16N_CNT_Typedef CNT;
 __IO T16N_CON0_Typedef CON0;
 __IO T16N_CON1_Typedef CON1;
 uint32_t RESERVED0 ;
 __IO T16N_PRECNT_Typedef PRECNT;
 __IO T16N_PREMAT_Typedef PREMAT;
 __IO T16N_IE_Typedef IE;
 __IO T16N_IF_Typedef IF;
 __IO T16N_MAT0_Typedef MAT0;
 __IO T16N_MAT1_Typedef MAT1;
 __IO T16N_MAT2_Typedef MAT2;
 __IO T16N_MAT3_Typedef MAT3;
} T16N_TypeDef;
typedef struct
{
 __IO T32N_CNT_Typedef CNT;
 __IO T32N_CON0_Typedef CON0;
 __IO T32N_CON1_Typedef CON1;
 uint32_t RESERVED0 ;
 __IO T32N_PRECNT_Typedef PRECNT;
 __IO T32N_PREMAT_Typedef PREMAT;
 __IO T32N_IE_Typedef IE;
 __IO T32N_IF_Typedef IF;
 __IO T32N_MAT0_Typedef MAT0;
 __IO T32N_MAT1_Typedef MAT1;
 __IO T32N_MAT2_Typedef MAT2;
 __IO T32N_MAT3_Typedef MAT3;
} T32N_TypeDef;
#define   APB_BASE   (0x40000000UL)
#define   T16N0_BASE   (APB_BASE + 0x02000)
#define   T16N1_BASE   (APB_BASE + 0x02400)
#define   T16N2_BASE   (APB_BASE + 0x02800)
#define   T16N3_BASE   (APB_BASE + 0x02C00)
#define   T32N0_BASE   (APB_BASE + 0x04000)
#define   T32N1_BASE   (APB_BASE + 0x04400)
#define T16N0 ((T16N_TypeDef *) T16N0_BASE )
#define T16N1 ((T16N_TypeDef *) T16N1_BASE )
#define T16N2 ((T16N_TypeDef *) T16N2_BASE )
```

```
#define T16N3 ((T16N_TypeDef *) T16N3_BASE )
#define T32N0 ((T32N_TypeDef *) T32N0_BASE )
#define T32N1 ((T32N_TypeDef *) T32N1_BASE )
#define T32N2 ((T32N_TypeDef *) T32N2_BASE )
```

6.3 宏定义

定时器/计数器的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_timer.h 中。

```
/* TIM 模块使能控制 */
#define T16N0_Enable() (T16N0->CON0.EN = 1)
#define T16N1_Enable() (T16N1->CON0.EN = 1)
#define T16N2_Enable() (T16N2->CON0.EN = 1)
#define T16N3_Enable() (T16N3->CON0.EN = 1)
#define T32N0_Enable() (T32N0->CON0.EN = 1)
#define T32N1_Enable() (T32N1->CON0.EN = 1)
#define T32N2_Enable() (T32N2->CON0.EN = 1)
#define T16N0_Disable() (T16N0->CON0.EN = 0)
#define T16N1_Disable() (T16N1->CON0.EN = 0)
#define T16N2_Disable() (T16N2->CON0.EN = 0)
#define T16N3_Disable() (T16N3->CON0.EN = 0)
#define T32N0_Disable() (T32N0->CON0.EN = 0)
#define T32N1_Disable() (T32N1->CON0.EN = 0)
#define T32N2_Disable() (T32N2->CON0.EN = 0)

/* 异步写使能控制 */
#define T16N0_ASYNCWR_Enable() (T16N0->CON0.ASYWEN = 1)
#define T16N1_ASYNCWR_Enable() (T16N1->CON0. ASYWEN = 1)
#define T16N2_ASYNCWR_Enable() (T16N2->CON0. ASYWEN= 1)
#define T16N3_ASYNCWR_Enable() (T16N3->CON0. ASYWEN= 1)
#define T32N0_ASYNCWR_Enable() (T32N0->CON0. ASYNCWREN = 1)
#define T32N1_ASYNCWR_Enable() (T32N1->CON0. ASYNCWREN = 1)
#define T32N2_ASYNCWR_Enable() (T32N2->CON0. ASYNCWREN= 1)
#define T16N0_ASYNCWR_Disable() (T16N0->CON0. ASYWEN= 0)
#define T16N1_ASYNCWR_Disable() (T16N1->CON0. ASYWEN = 0)
#define T16N2_ASYNCWR_Disable() (T16N2->CON0. ASYWEN = 0)
#define T16N3_ASYNCWR_Disable() (T16N3->CON0. ASYWEN = 0)
#define T32N0_ASYNCWR_Disable() (T32N0->CON0. ASYNCWREN= 0)
#define T32N1_ASYNCWR_Disable() (T32N1->CON0. ASYNCWREN = 0)
#define T32N2_ASYNCWR_Disable() (T32N2->CON0. ASYNCWREN = 0)

/* PWM 输出使能控制 */
#define T16N0_PwmOut0_Enable() (T16N0->CON1.MOE0 = 1)
#define T16N1_PwmOut0_Enable() (T16N1->CON1.MOE0 = 1)
#define T16N2_PwmOut0_Enable() (T16N2->CON1.MOE0 = 1)
```

```
#define T16N3_PwmOut0_Enable() (T16N3->CON1.MOE0 = 1)
#define T32N0_PwmOut0_Enable() (T32N0->CON1.MOE0 = 1)
#define T32N1_PwmOut0_Enable() (T32N1->CON1.MOE0 = 1)
#define T32N2_PwmOut0_Enable() (T32N2->CON1.MOE0 = 1)
#define T16N0_PwmOut1_Enable() (T16N0->CON1.MOE1 = 1)
#define T16N1_PwmOut1_Enable() (T16N1->CON1.MOE1 = 1)
#define T16N2_PwmOut1_Enable() (T16N2->CON1.MOE1 = 1)
#define T16N3_PwmOut1_Enable() (T16N3->CON1.MOE1 = 1)
#define T32N0_PwmOut1_Enable() (T32N0->CON1.MOE1 = 1)
#define T32N1_PwmOut1_Enable() (T32N1->CON1.MOE1 = 1)
#define T32N2_PwmOut1_Enable() (T32N2->CON1.MOE1 = 1)
#define T16N0_PwmOut0_Disable() (T16N0->CON1.MOE0 = 0)
#define T16N1_PwmOut0_Disable() (T16N1->CON1.MOE0 = 0)
#define T16N2_PwmOut0_Disable() (T16N2->CON1.MOE0 = 0)
#define T16N3_PwmOut0_Disable() (T16N3->CON1.MOE0 = 0)
#define T32N0_PwmOut0_Disable() (T32N0->CON1.MOE0 = 0)
#define T32N1_PwmOut0_Disable() (T32N1->CON1.MOE0 = 0)
#define T32N2_PwmOut0_Disable() (T32N2->CON1.MOE0 = 0)
#define T16N0_PwmOut1_Disable() (T16N0->CON1.MOE1 = 0)
#define T16N1_PwmOut1_Disable() (T16N1->CON1.MOE1 = 0)
#define T16N2_PwmOut1_Disable() (T16N2->CON1.MOE1 = 0)
#define T16N3_PwmOut1_Disable() (T16N3->CON1.MOE1 = 0)
#define T32N0_PwmOut1_Disable() (T32N0->CON1.MOE1 = 0)
#define T32N1_PwmOut1_Disable() (T32N1->CON1.MOE1 = 0)
#define T32N2_PwmOut1_Disable() (T32N2->CON1.MOE1 = 0)
```

6.4 库函数

定时器/计数器库函数定义于 lib_timer.c 中，声明于 lib_timer.h 中。

6.4.1 函数T16Nx_BaseInit

- 函数原型：void T16Nx_BaseInit(T16N_TypeDef* T16Nx,TIM_BaseInitStruType* TIM_BaseInitStruct)
- 功能描述：T16Nx 基本初始化
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ TIM_BaseInitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* TIM初始化配置结构体定义 */
typedef struct
```

```
{
    TIM_TYPE_CLKS   TIM_ClkS;          //时钟源选择
    TYPE_FUNCEN   TIM_SYNC;            //外部时钟同步
    TIM_TYPE_EDGE   TIM_EDGE;          //外部时钟计数边沿选择
    TIM_TYPE_MODE   TIM_Mode;          //工作模式选择
}TIM_BaseInitStruType;
```

时钟源选择枚举类型 TIM_TYPE_CLKS：

枚举元素	数值	描述
TIM_ClkS_PCLK	0x0	内部 PCLK
TIM_ClkS_CK0	0x1	外部 CK0 时钟输入
TIM_ClkS_CK1	0x2	外部 CK1 时钟输入

表 6-1　TIM_TYPE_CLKS

外部时钟计数边沿选择枚举类型 TIM_TYPE_EDGE：

枚举元素	数值	描述
TIM_EDGE_Rise	0x0	上升沿
TIM_EDGE_Fall	0x1	下降沿
TIM_EDGE_All	0x2	上升沿+下降沿

表 6-2　TIM_TYPE_EDGE

工作模式选择枚举类型 TIM_TYPE_MODE：

枚举元素	数值	描述
TIM_Mode_TC0	0x0	定时、计数模式
TIM_Mode_TC1	0x1	定时、计数模式
TIM_Mode_CAP	0x2	捕捉模式
TIM_Mode_PWM	0x3	调制模式

表 6-3　TIM_TYPE_MODE

6. 4. 2　函数T32Nx_BaseInit

- 函数原型：(T32N_TypeDef* T32Nx,TIM_BaseInitStruType* TIM_BaseInitStruct)
- 功能描述：T32Nx 基本初始化
- 输入参数：
  - T32Nx：可以是 T32N0/1/2
  - TIM_BaseInitStruct：初始化配置结构体地址
- 返回值：无

6. 4. 3 函数T16Nx_CapInit

- 函数原型：void T16Nx_CapInit(T16N_TypeDef* T16Nx,TIM_CapInitStruType* TIM_CapInitStruct)
- 功能描述：T16Nx 捕捉功能初始化函数
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ TIM_CapInitStruct：初始化配置结构体地址
- 返回值：无

6. 4. 4 函数T32Nx_CapInit

- 函数原型：void T32Nx_CapInit(T32N_TypeDef* T32Nx,TIM_CapInitStruType* TIM_CapInitStruct)
- 功能描述：T32Nx 捕捉初始化
- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
  - ◇ TIM_CapInitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* 捕捉功能初始化结构体定义 */
typedef struct
{
    TYPE_FUNCEN   TIM_CapRise;          //上升沿捕捉使能
    TYPE_FUNCEN   TIM_CapFall;          //下降沿捕捉使能
    TYPE_FUNCEN   TIM_CapIS0;           //输入端口0使能
    TYPE_FUNCEN   TIM_CapIS1;           //输入端口1使能
    TIM_TYPE_CAPT   TIM_CapTime;        //捕捉次数控制
 }TIM_CapInitStruType;
```

捕捉次数枚举类型 TIM_TYPE_CAPT：

枚举元素	数值	描述
TIM_CapTime_1	0x0	捕捉次数：1
TIM_CapTime_2	0x1	捕捉次数：2
TIM_CapTime_3	0x2	捕捉次数：3
TIM_CapTime_4	0x3	捕捉次数：4
TIM_CapTime_5	0x4	捕捉次数：5

枚举元素	数值	描述
TIM_CapTime_6	0x5	捕捉次数：6
TIM_CapTime_7	0x6	捕捉次数：7
TIM_CapTime_8	0x7	捕捉次数：8
TIM_CapTime_9	0x8	捕捉次数：9
TIM_CapTime_10	0x9	捕捉次数：10
TIM_CapTime_11	0XA	捕捉次数：11
TIM_CapTime_12	0XB	捕捉次数：12
TIM_CapTime_13	0XC	捕捉次数：13
TIM_CapTime_14	0XD	捕捉次数：14
TIM_CapTime_15	0xE	捕捉次数：15
TIM_CapTime_16	0xF	捕捉次数：16

表 6-4　TIM_TYPE_CAPT

6. 4. 5　函数T16Nx_MATxITConfig

- 函数原型：
  - void T16Nx_MAT0ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_MATCON Type)
  - void T16Nx_MAT1ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_MATCON Type)
  - void T16Nx_MAT2ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_MATCON Type)
  - void T16Nx_MAT3ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_MATCON Type)
- 功能描述：T16Nx 匹配后的计数模式及中断模式配置
- 输入参数：
  - T16Nx：可以是 T16N0/1/2/3
  - Type：匹配后的工作模式，详见表 6-5
- 返回值：无

6. 4. 6　函数T32Nx_MATxITConfig

- 函数原型：
  - void T32Nx_MAT0ITConfig(T32N_TypeDef* T32Nx,TIM_TYPE_MATCON Type)
  - void T32Nx_MAT1ITConfig(T32N_TypeDef* T32Nx,TIM_TYPE_MATCON Type)
  - void T32Nx_MAT2ITConfig(T32N_TypeDef* T32Nx,TIM_TYPE_MATCON Type)
  - void T32Nx_MAT3ITConfig(T32N_TypeDef* T32Nx,TIM_TYPE_MATCON Type)
- 功能描述：T32Nx 匹配后的计数模式及中断模式配置
- 输入参数：
  - T16Nx：可以是 T32N0/1
  - Type：匹配后的工作模式，详见表 6-5
- 返回值：无

匹配后的工作模式枚举类型 TIM_TYPE_MATCON：

枚举元素	数值	描述
TIM_Go_No	0x0	继续计数，不产生中断
TIM_Hold_Int	0x1	保持计数，产生中断
TIM_Clr_Int	0x2	清零并重新计数，产生中断
TIM_Go_Int	0x3	继续计数，产生中断

表 6-5　TIM_TYPE_MATCON

6. 4. 7　函数T16Nx_MATxOutxConfig

- 函数原型：
  - void T16Nx_MAT0Out0Config(T16N_TypeDef* T16Nx,TIM_TYPE_MATOUT Type)
  - void T16Nx_MAT1Out0Config(T16N_TypeDef* T16Nx,TIM_TYPE_MATOUT Type)
  - void T16Nx_MAT2Out1Config(T16N_TypeDef* T16Nx,TIM_TYPE_MATOUT Type)
  - void T16Nx_MAT3Out1Config(T16N_TypeDef* T16Nx,TIM_TYPE_MATOUT Type)
- 功能描述：T16Nx 匹配后的输出端口的模式配置
- 输入参数：无
  - T16Nx：可以是 T16N0/1/2/3
  - Type：匹配后的工作模式，详见表 6-6
- 返回值：无

6. 4. 8　函数T32Nx_MATxOutxConfig

- 函数原型：
  - void T32Nx_MAT0Out0Config(T32N_TypeDef* T32Nx,TIM_TYPE_MATOUT Type)
  - void T32Nx_MAT1Out0Config(T32N_TypeDef* T32Nx,TIM_TYPE_MATOUT Type)
  - void T32Nx_MAT2Out1Config(T32N_TypeDef* T32Nx,TIM_TYPE_MATOUT Type)
  - void T32Nx_MAT3Out1Config(T32N_TypeDef* T32Nx,TIM_TYPE_MATOUT Type)
- 功能描述：T32Nx 匹配后的输出端口的模式配置
- 输入参数：无
  - T32Nx：可以是 T32N0/1/2
  - Type：匹配后的工作模式，详见表 6-6
- 返回值：无

匹配后端口的工作模式枚举类型 TIM_TYPE_MATOUT：

枚举元素	数值	描述
TIM_Out_Hold	0x0	端口：保持
TIM_Out_Low	0x1	端口：清 0

枚举元素	数值	描述
TIM_Out_High	0x2	端口：置 1
TIM_Out_Switch	0x3	端口：取反

表 6-6　TIM_TYPE_MATOUT

6. 4. 9　函数T16Nx_ITConfig

- 函数原型：void T16Nx_ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_IT Type,TYPE_FUNCEN NewState)
- 功能描述：T16N 中断配置
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ Type：中断类型，详见表 6-7
  - ◇ NewState：使能/失能
- 返回值：无

6. 4. 10　函数T32Nx_ITConfig

- 函数原型：void T32Nx_ITConfig(T32N_TypeDef* T32Nx,TIM_TYPE_IT Type,TYPE_FUNCEN NewState)
- 功能描述：T32N 中断配置
- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
  - ◇ Type：中断类型，详见表 6-7
  - ◇ NewState：使能/失能
- 返回值：无

中断配置枚举类型 TIM_TYPE_IT：

枚举元素	数值	描述
TIM_IT_MAT0	0x01	匹配 0 中断
TIM_IT_MAT1	0x02	匹配 1 中断
TIM_IT_MAT2	0x04	匹配 2 中断
TIM_IT_MAT3	0x08	匹配 3 中断
TIM_IT_N	0x10	T16N 匹配 0xFFFF/ T32N 匹配 0xFFFFFFFF 中断
TIM_IT_CAP0	0x20	输入端口 0 捕捉中断
TIM_IT_CAP1	0x40	输入端口 1 捕捉中断

表 6-7　TIM_TYPE_IT

6. 4. 11 函数T16Nx_PWMOutConfig

- 函数原型：void T16Nx_PWMOutConfig(TIM_TYPE_PWMOC Pwms,PWM_TYPE_OUT PwmType,TYPE_LEVEL PwmLevel)
- 功能描述：T16N0OUT、T16N1OUT、T16N2OUT、T16N3OUT 输出配置
- 输入参数：
  - Pwms：可以是 T16N0OUT/T16N1OUT/T16N2OUT/T16N3OUT，详见表 6-8
  - PwmType 可以是 NO_PWM/ IR38K_PWM/ PWM_OUT0/ PWM_OUT1
  - NewState：使能/失能
- 返回值：无

注意：此功能为 UART TX 脉宽调制输出功能的扩展，因此不能与相关的 UART 共用。

枚举元素	数值	描述
T16N0_OUT	0x00	T16N0 OUT 端口 / TX0
T16N1_OUT	0x01	T16N1 OUT 端口 / TX1
T16N2_OUT	0x02	T16N2 OUT 端口 / TX2
T16N3_OUT	0x03	T16N3 OUT 端口 / TX3

表 6-8 TIM_TYPE_PWMOC

枚举元素	数值	描述
NO_PWM	0x00	非调制输出
IR38K_PWM	0x01	固定 38K 调制输出
PWM_OUT0	0x02	T16NxOUT0 PWM 调制输出
PWM_OUT1	0x03	T16NxOUT1 PWM 调制输出

表 6-9 PWM_TYPE_OUT

6. 4. 12 函数T16Nx_SetCNT

- 函数原型：void T16Nx_SetCNT(T16N_TypeDef* T16Nx,uint16_t Value)
- 功能描述：设置计数值
- 输入参数：
  - T16Nx：可以是 T16N0/1/2/3
  - Value：16 位数值
- 返回值：无

6. 4. 13 函数T32Nx_SetCNT

- 函数原型：void T32Nx_SetCNT(T32N_TypeDef* T32Nx,uint32_t Value)

- 功能描述：设置计数值
- 输入参数：
  - T32Nx：可以是 T16N0/1/2
  - Value： 32 位数值（16 位时右对齐）
- 返回值：无

6. 4. 14 函数T16Nx_SetPRECNT

- 函数原型：void T16Nx_SetPRECNT(T16N_TypeDef* T16Nx, uint8_t Value)
- 功能描述：设置预分频计数寄存器值
- 输入参数：
  - T16Nx：可以是 T16N0/1/2/3
  - Value：8 位数值
- 返回值：无

6. 4. 15 函数T32Nx_SetPRECNT

- 函数原型：void T32Nx_SetPRECNT(T32N_TypeDef* T32Nx, uint8_t Value)
- 功能描述：设置预分频计数寄存器值
- 输入参数：
  - T32Nx：可以是 T32N0/1/2/3
  - Value：8 位数值
- 返回值：无

6. 4. 16 函数T16Nx_SetPREMAT

- 函数原型：void T16Nx_SetPREMAT (T16N_TypeDef* T16Nx, uint8_t Value)
- 功能描述：设置预分频计数匹配寄存器值
- 输入参数：
  - T16Nx：可以是 T16N0/1/2/3
  - Value：8 位数值
- 返回值：无

6. 4. 17 函数T32Nx_SetPREMAT

- 函数原型：void T32Nx_SetPREMAT (T32N_TypeDef* T32Nx, uint8_t Value)

- 功能描述：设置预分频计数匹配寄存器值
- 输入参数：
  - T32Nx：可以是 T32N0/1/2
  - Value：8 位数值
- 返回值：无

6. 4. 18 函数T16Nx_SetMAT0、T16Nx_SetMAT1、T16Nx_SetMAT2、T16Nx_SetMAT3

- 函数原型：T16Nx_SetMAT0(T16N_TypeDef* T16Nx,uint16_t Value)

  T16Nx_SetMAT1(T16N_TypeDef* T16Nx,uint16_t Value)

  T16Nx_SetMAT2(T16N_TypeDef* T16Nx,uint16_t Value)

  T16Nx_SetMAT3(T16N_TypeDef* T16Nx,uint16_t Value)

- 功能描述：设置匹配寄存器
- 输入参数：
  - T16Nx：可以是 T16N0/1/2/3
  - Value：16
- 返回值：无

6. 4. 19 函数T32Nx_SetMAT0、T32Nx_SetMAT1、T32Nx_SetMAT2、T32Nx_SetMAT3

- 函数原型：T32Nx_SetMAT0(T32N_TypeDef* T32Nx,uint32_t Value)

  T32Nx_SetMAT1(T32N_TypeDef* T32Nx,uint32_t Value)

  T32Nx_SetMAT2(T32N_TypeDef* T32Nx,uint32_t Value)

  T32Nx_SetMAT3(T32N_TypeDef* T32Nx,uint32_t Value)

- 功能描述：设置匹配寄存器
- 输入参数：
  - T32Nx：可以是 T32N0/1/2
  - Value：32 位数值（16 位时右对齐）
- 返回值：无

6. 4. 20 函数T16Nx_GetMAT0、T16Nx_GetMAT1、T16Nx_GetMAT2、T16Nx_GetMAT3

- 函数原型：uint16_t T16Nx_GetMAT0(T16N_TypeDef* T16Nx)

  uint16_t T16Nx_GetMAT1(T16N_TypeDef* T16Nx)

  uint16_t T16Nx_GetMAT2(T16N_TypeDef* T16Nx)

  uint16_t T16Nx_GetMAT3(T16N_TypeDef* T16Nx)
- 功能描述：读取匹配寄存器值
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
- 返回值：16 位数值

6. 4. 21 函数T32Nx_GetMAT0、T32Nx_GetMAT1、T32Nx_GetMAT2、T32Nx_GetMAT3

- 函数原型：uint32_t T32Nx_GetMAT0(T32N_TypeDef* T32Nx)

  uint32_t T32Nx_GetMAT1(T32N_TypeDef* T32Nx)

  uint32_t T32Nx_GetMAT2(T32N_TypeDef* T32Nx)

  uint32_t T32Nx_GetMAT3(T32N_TypeDef* T32Nx)
- 功能描述：读取匹配寄存器值
- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
- 返回值： 32 位数值（16 位时右对齐）

6. 4. 22 函数T16Nx_GetCNT

- 函数原型：uint16_t T16Nx_GetCNT(T16N_TypeDef* T16Nx)
- 功能描述：读取计数寄存器值
- 输入参数：可以是 T16N0/1/2/3
- 返回值：16 位数值

6. 4. 23 函数T32Nx_GetCNT

- 函数原型：uint32_t T32Nx_GetCNT(T32N_TypeDef* T32Nx)
- 功能描述：读取计数寄存器值
- 输入参数：可以是 T32N0/1/2
- 返回值：32 位数值（16 位时右对齐）

6. 4. 24 函数T16Nx_GetPRECNT

- 函数原型：uint8_t T16Nx_GetPRECNT(T16N_TypeDef* T16Nx)
- 功能描述：读取预分频计数寄存器值
- 输入参数：可以是 T16N0/1/2/3
- 返回值：8 位数值

6. 4. 25 函数T32Nx_GetPRECNT

- 函数原型：uint8_t T32Nx_GetPRECNT(T32N_TypeDef* T32Nx)
- 功能描述：读取预分频计数寄存器值
- 输入参数：可以是 T32N0/1/2
- 返回值：8 位数值

6. 4. 26 函数T16Nx_GetITStatus

- 函数原型：ITStatus T16Nx_GetITStatus(T16N_TypeDef* T16Nx,TIM_TYPE_IT TIM_Flag)
- 功能描述：读取指定标志位
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：RESET/SET

6. 4. 27 函数T32Nx_GetFlagStatus

- 函数原型：FlagStatus T32Nx_GetFlagStatus(T32N_TypeDef* T32Nx,TIM_TYPE_IT TIM_Flag)
- 功能描述：读取指定标志位

- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：RESET/SET

6. 4. 28 函数T16Nx_GetITStatus

- 函数原型：ITStatus T16Nx_GetITStatus(T16N_TypeDef* T16Nx, TIM_TYPE_IT TIM_Flag)
- 功能描述：读取指定中断状态,未使能相应中断时不会返回 SET
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：RESET/SET

6. 4. 29 函数T32Nx_GetITStatus

- 函数原型：ITStatus T32Nx_GetITStatus(T32N_TypeDef* T32Nx, TIM_TYPE_IT TIM_Flag)
- 功能描述：读取指定中断状态,未使能相应中断时不会返回 SET
- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：RESET/SET

6. 4. 30 函数T16Nx_ClearITPendingBit

- 函数原型：T16Nx_ClearITPendingBit(T16N_TypeDef* T16Nx,TIM_TYPE_IT TIM_Flag)
- 功能描述：清除指定的中断标志位
- 输入参数：
  - ◇ T16Nx：可以是 T16N0/1/2/3
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：无

6. 4. 31 函数T32Nx_ClearITPendingBit

- 函数原型：T32Nx_ClearITPendingBit(T32N_TypeDef* T32Nx,TIM_TYPE_IT TIM_Flag)
- 功能描述：清除指定的中断标志位
- 输入参数：
  - ◇ T32Nx：可以是 T32N0/1/2
  - ◇ TIM_Flag：中断标志位，详见表 6-7
- 返回值：无

6. 5 函数库应用示例

```
/* T16N3定时器初始化：5ms定时中断 */
void User_T16N3Init(void)
{
    TIM_BaseInitStruType x;                                  //定义结构
    x.TIM_ClkS = TIM_ClkS_PCLK;                              //选择时钟源
    x.TIM_Mode = TIM_Mode_TC0 ;                              //选择工作模式
    T16Nx_BaseInit(T16N3,&x);                                //初始化定时器
    T16Nx_SetPREMAT(T16N3,1);                                //预分频1:2
    T16Nx_SetMAT0(T16N3,40000);                              //设置匹配寄存器
    NVIC_Init(NVIC_T16N3_IRQn,NVIC_Priority_1,ENABLE);       //开总中断
    T16Nx_MAT0ITConfig(T16N3,TIM_Clr_Int);                   //MAT0匹配中断
    T16Nx_ITConfig(T16N3,TIM_IT_MAT0,ENABLE);                //使能MAT0中断
    T16N3_Enable();                                          //使能T16N3
}
/* T16N3定时器初始化：5ms定时中断 */
void T16N3_IRQHandler(void)
{
    if(T16Nx_GetFlagStatus(T16N3,TIM_IT_MAT0) != RESET)
//判断中断类型
    {
        T16Nx_ClearITPendingBit(T16N3,TIM_IT_MAT0);          //清除标志
        UserFunction();                                      //用户程序
    }
}
```

第7章 模拟/数字转换器（ADC）

7.1 功能概述

- 支持 12 位采样精度
- 支持 11 个模拟输入端
- 支持 12 位转换结果，可选择高位对齐或低位对齐格式
- 支持 ADC 中断标志 IF，可唤醒睡眠模式
- 支持正负向参考电压可配置
- 支持转换时钟可配置

7.2 寄存器结构

ADC 模块的寄存器定义于文件 ES8H296.h。芯片支持 1 个模数转换器。

```
typedef struct
{
 __I ADC_DR_Typedef DR;
 __IO ADC_CON0_Typedef CON0;
 __IO ADC_CON1_Typedef CON1;
 __IO ADC_CHS_Typedef CHS;
 __IO ADC_IE_Typedef IE;
 __IO ADC_IF_Typedef IF;
 uint32_t RESERVED0[2] ;
 __IO ADC_BUF_Typedef BUF;
} ADC_TypeDef;

#define   APB_BASE   (0x40000000UL)
#define   ADC_BASE   (APB_BASE + 0x01000)
#define   ADC   ((ADC_TypeDef *) ADC_BASE )
```

7.3 宏定义

ADC 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_adc.h 中。

```
/* ADC使能控制 */
#define ADC_Enable()   (ADC->CON0.EN = 0x1)
#define ADC_Disable() (ADC->CON0.EN = 0x0)

/* ADC开始转换 */
#define ADC_Start() (ADC->CON0.TRG = 0x1)

/* ADC采样软件控制 */
```

```
#define ADC_SampStart() (ADC->CON1.SMPON = 0x1)
#define ADC_SampStop()   (ADC->CON1.SMPON = 0x0)

/*  清除中断标志  */
#define ADC_ClearITPendingBit() (ADC->IF.IF = 0x1)

/* ADC中断使能控制  */
#define ADC_IT_Enable()   (ADC->IE.IE = 0x1)
#define ADC_IT_Disable() (ADC->IE.IE = 0x0)
```

7. 4 库函数

ADC 库函数定义于 lib_adc.c 中，声明于 lib_adc.h 中。

7. 4. 1 函数ADC_Init

- 函数原型：void ADC_Init(ADC_InitStruType * ADC_InitStruct)
- 功能描述：初始化 ADC 模块
- 输入参数：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/*ADC初始化配置结构体定义*/
typedef struct
{
    ADC_TYPE_CLKS   ADC_ClkS;          //ADC时钟源选择
    ADC_TYPE_CLKDIV   ADC_ClkDiv;      //ADC时钟源预分频
    ADC_TYPE_FM   ADC_Align;           //ADC结果对齐方式
    ADC_TYPE_VREFP   ADC_VrefP;        //ADC正向参考电压选择
    ADC_TYPE_SMPS   ADC_SampS;         //ADC采样模式选择
    ADC_TYPE_ST   ADC_SampClk;         //ADC采样时间选择
    ADC_TYPE_HSEN   ADC_ConvSpeed;     //ADC转换速度选择
    ADC_TYPE_CHS   ADC_ChS;            //ADC模拟通道选择
     ADC_TYPE_BUFLP   ADC_Lp;          //ADC_BUF_LP    ADC core功耗选择

   ADC_TYPE_BUFEN ADC_Buf;             //ADC_BUF_EN    ADC输入buffer选择
}ADC_InitStruType;
```

ADC 时钟源选择枚举类型 ADC_TYPE_CLKS：

枚举元素	数值	描述
ADC_ClkS_Pclk	0x0	ADC 时钟源选择：PCLK
ADC_ClkS_32Khz	0x1	ADC 时钟源选择：ADCCLK(32KHZ)

表 7-1 ADC_TYPE_CLKS

ADC 时钟源预分频枚举类型 ADC_TYPE_CLKDIV：

枚举元素	数值	描述
ADC_ClkDiv_1	0x0	预分频：1:1
ADC_ClkDiv_2	0x1	预分频：1:2
ADC_ClkDiv_4	0x2	预分频：1:4
ADC_ClkDiv_8	0x3	预分频：1:8
ADC_ClkDiv_16	0x4	预分频：1:16
ADC_ClkDiv_32	0x5	预分频：1:32
ADC_ClkDiv_64	0x6	预分频：1:64
ADC_ClkDiv_256	0x7	预分频：1:256

表 7-2　ADC_TYPE_CLKDIV

ADC 结果对齐方式枚举类型 ADC_TYPE_FM：

枚举元素	数值	描述
ADC_Align_Right	0x1	结果对齐：右对齐（低对齐）
ADC_Align_Left	0x0	结果对齐：左对齐（高对齐）

表 7-3　ADC_TYPE_FM

正向参考电压选择枚举类型 ADC_TYPE_VREFP：

枚举元素	数值	描述
ADC_VrefP_Vcc	0x0	正向参考电压：3.3V LDO 电压
ADC_VrefP_Ref	0x1	正向参考电压：内部参考电压 Vref 2.5V ，AVREFP 端口复用为普通 IO 口
ADC_VrefP_RefOut	0x2	正向参考电压：内部参考电压 Vref 2.5V ，AVREFP 端口输出 Vref
ADC_VrefP_Exter	0x3	正向参考电压：外部参考电压

表 7-4　ADC_TYPE_VREFP

采样模式选择枚举类型 ADC_TYPE_SMPS：

枚举元素	数值	描述
ADC_Samp_Soft	0x0	采样模式选择：软件
ADC_Samp_Hardware	0x1	采样模式选择：硬件

表 7-5　ADC_TYPE_SMPS

采样时间选择枚举类型 ADC_TYPE_ST：

枚举元素	数值	描述
ADC_SampClk_2	0x0	采样时间选择：2 个 TadClk
ADC_SampClk_4	0x1	采样时间选择：4 个 TadClk

枚举元素	数值	描述
ADC_SampClk_8	0x2	采样时间选择：8 个 TadClk
ADC_SampClk_16	0x3	采样时间选择：16 个 TadClk

表 7-6　ADC_TYPE_ST

转换速度选择枚举类型 ADC_TYPE_HSEN：

枚举元素	数值	描述
ADC_ConvSpeed_Low	0x0	转换速度：低速
ADC_ConvSpeed_High	0x1	转换速度：高速

表 7-7　ADC_TYPE_HSEN

ADC 通道选择枚举类型 ADC_TYPE_CHS：

枚举元素	数值	描述
ADC_Ch_0	0x0	通道 0
ADC_Ch_1	0x1	通道 1
ADC_Ch_2	0x2	通道 2
ADC_Ch_3	0x3	通道 3
ADC_Ch_4	0x4	通道 4
ADC_Ch_5	0x5	通道 5
ADC_Ch_6	0x6	通道 6
ADC_Ch_7	0x7	通道 7
ADC_Ch_8	0x8	通道 8
ADC_Ch_9	0x9	通道 9
ADC_Ch_10	0Xa	通道 10

表 7-8　ADC_TYPE_CHS

ADC core 功耗选择位 ADC_TYPE_BUFLP：

枚举元素	数值	描述
ADC_LP_86ua	0x2	ADC Core 功耗为 86，推荐使用
ADC_LP_86ua	0x0	ADC Core 功耗为 86，推荐使用
ADC_LP_86ua	0x1	采样时间选择：8 个 TadClk
ADC_LP_86ua	0x3	ADC Core 功耗为 86，推荐使用

表 7-9　ADC_TYPE_BUFLP

ADC 输入 buff 功耗选择位 ADC_TYPE_ST：

枚举元素	数值	描述
ADC_BUFF_0ua	0x0	ADC 输入 buff 功耗为 0uA，推荐使用
ADC_BUFF_34ua	0x1	ADC 输入 buff 功耗为 34uA
ADC_BUFF_95ua	0x2	ADC 输入 buff 功耗为 95uA

表 7-10　ADC_TYPE_BUFEN

7. 4. 2 函数ADC_Set_CH

- 函数原型：void ADC_Set_CH(ADC_TYPE_CHS AdcCH)
- 功能描述：选择 ADC 模拟通道
- 输入参数：ADC 模拟通道，详见表 7-8
- 返回值：无

7. 4. 3 函数ADC_GetConvValue

- 函数原型：uint16_t ADC_GetConvValue(void)
- 功能描述：获取 ADC 转换结果
- 输入参数：无
- 返回值：转换值

7. 4. 4 函数ADC_GetConvStatus

- 函数原型：FlagStatus ADC_GetConvStatus(void)
- 功能描述：获取 ADC 转换状态
- 输入参数：无
- 返回值：RESET(完成)/SET(正在转换)

7. 4. 5 函数ADC_GetFlagStatus

- 函数原型：FlagStatus ADC_GetFlagStatus(void)
- 功能描述：读取 ADC 转换完成中断标志
- 输入参数：无
- 返回值：RESET(完成)/SET(正在转换)

7. 4. 6 函数ADC_GetITStatus

- 函数原型：ITStatus ADC_GetITStatus(void)
- 功能描述：获取 ADC 中断状态，未使能相应中断时不会返回 SET
- 输入参数：无
- 返回值：SET（中断）/RESET（无中断）

7. 4. 7 函数ADC_Reset

- 函数原型：void ADC_Reset(void)
- 功能描述：ADC 模块重置，寄存器恢复上电初始值
- 输入参数：无
- 返回值：无

7. 5 库函数应用示例

```
/* 初始化ADC函数 */
void  ADC_UserInit(void)
{
    ADC_InitStruType ADC_InitStruct;                          //定义结构体
    ADC_InitStruct.ADC_ClkS = ADC_ClkS_Pclk;                  //时钟
    ADC_InitStruct.ADC_ClkDiv = ADC_ClkDiv_256;               //预分频
    ADC_InitStruct.ADC_Align = ADC_Align_Left;                //结果对齐
    ADC_InitStruct.ADC_VrefP = ADC_VrefP_Ref;                 //正向参考电压
    ADC_InitStruct.ADC_ConvSpeed = ADC_ConvSpeed_Low;         //AD转换速度
    ADC_InitStruct.ADC_SampS = ADC_Samp_Hardware;             //AD采样模式
    ADC_InitStruct.ADC_SampClk = ADC_SampClk_16;              //AD采样时间
    ADC_InitStruct.ADC_Buf = ADC_BUFF_0ua;                    //通道
    ADC_InitStruct.ADC_Lp = ADC_LP_86ua;
    ADC_Init(&ADC_InitStruct);                                //配置ADC
    ADC_IT_Disable();                                         //不使能中断
    ADC_Enable();                                             //使能ADC
}
    /* 主函数 */
int main(void)
{
    Uint16_t  Temp16;
    SystemClockConfig();                                      //配置时钟
    SCU_ADCCLK_Enable();                                      // ADC时钟使能
    GPIO_UserInit();                                          //GPIO初始化
    ADC_UserInit();                                           //初始化ADC
    while(1)
    {
        ADC_Start();                                          //ADC转换开始
        while(ADC_GetConvStatus()==SET);                      //等待转换完成
        ADC_ClearITPendingBit;                                //清除标志位
        Temp16 = ADC_GetConvValue();                          //读取转换值
        UserFunction();                                       //用户程序
    }
}
```

第8章 液晶显示控制器（LCDC）

8. 1 功能概述

- 支持 8COM x 56SEG
- 支持灰度调节功能
- 支持显示闪烁功能，闪烁频率可调

8. 2 寄存器结构

LCDC 的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
 __IO LCDC_CON0_Typedef CON0;
 __IO LCDC_FLKT_Typedef FLKT;
 __IO LCDC_SEL0_Typedef SEL0;
 __IO LCDC_SEL1_Typedef SEL1;
 __IO LCDC_CON1_Typedef CON1;
 __IO LCDC_IE_Typedef IE;
 __IO LCDC_IF_Typedef IF;
 uint32_t RESERVED0 ;
 __IO LCDC_D0_Typedef D0;
 __IO LCDC_D1_Typedef D1;
 __IO LCDC_D2_Typedef D2;
 __IO LCDC_D3_Typedef D3;
 __IO LCDC_D4_Typedef D4;
 __IO LCDC_D5_Typedef D5;
 __IO LCDC_D6_Typedef D6;
 __IO LCDC_D7_Typedef D7;
 __IO LCDC_D8_Typedef D8;
 __IO LCDC_D9_Typedef D9;
 __IO LCDC_D10_Typedef D10;
 __IO LCDC_D11_Typedef D11;
} LCDC_TypeDef;
#define APB_BASE (0x40000000UL)
#define LCDC_BASE (APB_BASE + 0x01800)
#define LCDC ((LCDC_TypeDef *) LCDC_BASE )
```

8. 3 宏定义

LCDC 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_lcd.h 中。

```
/* LCD驱动使能控制  */
```

```
#define LCD_Enable()   (LCDC->CON1.EN= 0x1)
#define LCD_Disable() (LCDC->CON1.EN= 0x0)
/* LCD显示闪烁使能控制 */
#define LCD_FLIK_Enable()   (LCDC->CON0.FLIK = 0x1)
#define LCD_FLIK_Disable() (LCDC->CON0.FLIK = 0x0)
/* 像素寄存器刷新请求 */
#define LCD_SetRFREQ()    (LCDC->CON1.RFREQ = 0x1)
#define LCD_ClearRFREQ() (LCDC->CON1.RFREQ = 0x0)
/* 清LCD驱动关闭中断标志 */
#define LCD_ClearOFFIF() ( LCDC->IF.OFFIF = 0x1)
/* 清LCD像素寄存器刷新中断标志 */
#define LCD_ClearRFIF() (LCDC->IF.RFIF = 0x1)
/* LCD软件复位 */
#define LCD_Rest() (LCDC->CON1.RST = 0x01)
```

8. 4 库函数

LCDC 库函数定义于 lib_lcd.c 中，声明于 lib_lcd.h 中。

8. 4. 1 函数LCD_Init

- 函数原型：ErrorStatus LCD_Init(LCD_InitStruType * LCD_InitStruct,LCDSEL_TYPE SELx)
- 功能描述：初始化 LCD 模块
- 输入参数：初始化配置结构体地址, SELx:选择的 LCD 段
- 返回值：成功、失败

初始化配置结构原型：

```
/* LCD初始化配置结构体定义 */
typedef struct
{
    LCD_TYPE_COMS   LCD_Coms;      // LCDC:bit2~0 公共端选择位
    TYPE_FUNCEN   LCD_LcdFlik;     // LCDC:bit5 LCD显示闪烁使能位
    LCD_TYPE_WFS   LCD_LcdWFS;     // LCDC:bit6 LCD驱动波形类型选择位
    LCD_TYPE_BIAS   LCD_Bias;      // LCDC:bit9~8 LCD偏置选择
    LCD_TYPE_LCDRS   LCD_RS;       // LCDC:bit11~10 LCD内部偏置电阻选择
    LCD_TYPE_BVS   LCD_Grayscale;  // LCDC:bit14~12 LCD显示灰度控制
    uint8_t   LCD_PRS;             // LCD时钟源分频比选择位 0~63
    uint32_t   LCD_SEG0;           // LCDSEL0 bit31~0 LCD段使能
    uint32_t   LCD_SEG0;           // LCDSEL1 bit15~0 LCD段使能
 }LCD_InitStruType;
```

公共端选择枚举类型 LCD_TYPE_COMS：

枚举元素	数值	描述
LCD_Coms_1	0x0	静态(COM0)
LCD_Coms_2	0x4	1/2(COM1~COM0)
LCD_Coms_3	0x6	1/3(COM2~COM0)
LCD_Coms_4	0x1	1/4(COM3~COM0)
LCD_Coms_6	0x2	1/6(COM5~COM0)
LCD_Coms_8	0x3	1/8(COM7~COM0)

表 8-1　LCD_TYPE_COMS

驱动波形枚举类型 LCD_TYPE_WFS：

枚举元素	数值	描述
LCD_WaveMode_A	0x0	A 型波形
LCD_WaveMode_B	0x1	B 型波形

表 8-2　LCD_TYPE_WFS

偏置选择枚举类型 LCD_TYPE_BIAS：

枚举元素	数值	描述
LCD_Bias_3	0x0	1/3 偏置
LCD_Bias_4	0x3	1/4 偏置

表 8-3　LCD_TYPE_BIAS

内部偏置电阻枚举类型 LCD_TYPE_LCDRS ：

枚举元素	数值	描述
LCD_Res_15k	0x0	偏压电阻 15k
LCD_Res_50k	0x1	偏压电阻 50k
LCD_Res_100k	0x2	偏压电阻 100k
LCD_Res_200k	0x3	偏压电阻 200k

表 8-4　LCD_TYPE_LCDRS

显示灰度枚举类型 LCD_TYPE_BVS：

枚举元素	数值	描述
LCD_Grayscale_VLCD	0x0	Vbias = VLCD
LCD_Grayscale_35_36_VLCD	0x1	Vbias = (35/36)/VLCD
LCD_Grayscale_34_36_VLCD	0x2	Vbias = (34/36)/VLCD
LCD_Grayscale_32_36_VLCD	0x3	Vbias = (32/36)/VLCD
LCD_Grayscale_31_36_VLCD	0x4	Vbias = (31/36)/VLCD
LCD_Grayscale_30_36_VLCD	0x5	Vbias = (30/36)/VLCD

枚举元素	数值	描述
LCD_Grayscale_29_36_VLCD	0x6	Vbias = (29/36)/VLCD
LCD_Grayscale_28_36_VLCD	0x7	Vbias = (28/36)/VLCD

表 8-5 LCD_TYPE_BVS

8. 4. 2 函数LCD_ITConfig

- 函数原型：void LCD_ITConfig(LCD_TYPE_IE LCD_IT, TYPE_FUNCEN NewState)
- 功能描述：LCD 中断配置
- 输入参数：
  - ◇ LCD_IT：需要配置的中断，详见表 8-6
  - ◇ NewState：使能或关闭
- 返回值：无

中断使能枚举类型 LCD_TYPE_IE：

枚举元素	数值	描述
LCD_IE_OFF	0x1	LCD 驱动关闭中断使能
LCD_IE_RF	0x2	LCD 像素寄存器刷新中断使能

表 8-6 LCD_TYPE_IE

8. 4. 3 函数LCD_GetRFITStatus

- 函数原型：ITStatus LCD_GetRFITStatus(void)
- 功能描述：读取 LCD 像素刷新中断标志
- 输入参数：无
- 返回值：SET/RESET

8. 4. 4 函数LCD_GetOFFITStatus

- 函数原型：ITStatus LCD_GetOFFITStatus(void)
- 功能描述：读取 LCD 驱动关闭中断标志
- 输入参数：无
- 返回值：SET/RESET

8. 4. 5 函数LCD_GrayscaleConfig

- 函数原型：void LCD_GrayscaleConfig(LCD_TYPE_BVS LCD_Grayscale)
- 功能描述：LCD 灰度选择

- 输入参数：灰度电压值，详见表 8-5
- 返回值：无

8. 4. 6 函数LCD_FlickerTimeConfig

- 函数原型：ErrorStatus LCD_FlickerTimeConfig(uint8_t On_Time,uint8_t Off_Time)
- 功能描述：闪烁时间设置
- 输入参数：
  - On_Time：闪烁时间/0.25s
  - Off_Time：熄灭时间/0.25s
- 返回值：成功、失败

8. 4. 7 函数LCD_PixelWriteByte

- 函数原型：ErrorStatus LCD_PixelWriteByte(LCD_TYPE_PIXEL LCD_DD, LCD_DD_BYTE nByte, uint8_t LCD_data)
- 功能描述：LCD 像素寄存器写字节数据
- 输入参数：
  - LCD_DD：LCD 像素寄存器名称，详见表 8-7
  - nByte：该像素寄存器的第几个字节，详见表 8-8
  - LCD_data：对应字节数据
- 返回值：成功、失败

像素寄存器名称枚举类型 LCD_TYPE_PIXEL：

枚举元素	数值	描述
LCD_Pixel_LCDD0	0x0	像素寄存器 D0
LCD_Pixel_LCDD1	0x1	像素寄存器 D1
LCD_Pixel_LCDD2	0x2	像素寄存器 D2
LCD_Pixel_LCDD3	0x3	像素寄存器 D3
LCD_Pixel_LCDD4	0x4	像素寄存器 D4
LCD_Pixel_LCDD5	0x5	像素寄存器 D5
LCD_Pixel_LCDD6	0x6	像素寄存器 D6
LCD_Pixel_LCDD7	0x7	像素寄存器 D7
LCD_Pixel_LCDD8	0x8	像素寄存器 D8
LCD_Pixel_LCDD9	0x9	像素寄存器 D9
LCD_Pixel_LCDD10	0xA	像素寄存器 D10
LCD_Pixel_LCDD11	0xB	像素寄存器 D11
LCD_Pixel_LCDD12	0xC	像素寄存器 D12
LCD_Pixel_LCDD13	0xD	像素寄存器 D13

表 8-7 LCD_TYPE_PIXEL

像素寄存器字节选择枚举类型 LCD_DD_BYTE：

枚举元素	数值	描述
LCD_Byte_0	0x0	像素寄存器 Bit0-Bit7
LCD_Byte_1	0x1	像素寄存器 Bit8-Bit15
LCD_Byte_2	0x2	像素寄存器 Bit16-Bit23
LCD_Byte_3	0x3	像素寄存器 Bit24-Bit31

表 8-8　LCD_DD_BYTE

8. 4. 8　函数LCD_PixelWriteHalfWord

- 函数原型：ErrorStatus　LCD_PixelWriteByte(LCD_TYPE_PIXEL　LCD_DD, LCD_DD_BYTE nByte, uint8_t LCD_data)
- 功能描述：LCD 像素寄存器写 1 字节数据
- 输入参数：
  - ◇ LCD_DD：LCD 像素寄存器名称，详见表 8-7
  - ◇ nByte：该像素寄存器的第几个字节，详见表 8-10
  - ◇ LCD_data：对应字节数据
- 返回值：成功、失败

像素寄存器半字选择枚举类型 LCD_DD_BYTE：

枚举元素	数值	描述
LCD_Byte_0	0x0	像素寄存器 Bit0-Bit7
LCD_Byte_1	0x1	像素寄存器 Bit8-Bit15
LCD_Byte_2	0x2	像素寄存器 Bit16-Bit23
LCD_Byte_3	0x3	像素寄存器 Bit24-Bit31

表 8-9　LCD_DD_BYTE

8. 4. 9　函数LCD_PixelWriteHalfWord

- 函数原型：ErrorStatus　LCD_PixelWriteHalfWord(LCD_TYPE_PIXEL　LCD_DD, LCD_DD_HALFWORD　nHalfWord,　uint16_t　LCD_data)
- 功能描述：LCD 像素寄存器写半字数据
- 输入参数：
  - ◇ LCD_DD：LCD 像素寄存器名称，详见表 8-7
  - ◇ nHalfWord：该像素寄存器的第几个半字，详见表 8-10
  - ◇ LCD_data：对应半字数据
- 返回值：成功、失败

像素寄存器半字选择枚举类型 LCD_DD_HALFWORD：

枚举元素	数值	描述
LCD_HalfWord_0	0x0	像素寄存器 Bit0-Bit15
LCD_HalfWord_1	0x1	像素寄存器 Bit16-Bit31

表 8-10　LCD_DD_HALFWORD

8. 4. 10　函数LCD_PixelWriteWord

- 函数原型：ErrorStatus　LCD_PixelWriteWord(LCD_TYPE_PIXEL　LCD_DD, uint32_t　LCD_data)
- 功能描述：LCD 像素寄存器写字数据
- 输入参数：
  - LCD_DD：LCD 像素寄存器名称，详见表 8-7
  - LCD_data：对应字数据
- 返回值：成功、失败

8. 5　函数库应用示例

```
void User_LcdConfig(void)
{
    LCD_InitStruType sparam;                          //定义结构体
    SCU_LCDCCLK_Enable();                             //LCD时钟使能
    sparam.LCD_Coms =   LCD_Coms_4;                   //公共端选择com0~com3
    sparam.LCD_LcdFlik = DISABLE;                     //闪烁禁止
    sparam.LCD_LcdWFS = LCD_WaveMode_A;               //A型波型
    sparam.LCD_Bias = LCD_Bias_3;                     //1/3电压偏置
    sparam.LCD_RS = LCD_Res_50k;                      //50k偏压电阻
    sparam.LCD_Grayscale = LCD_Grayscale_3v2;         //3.2v灰度
    sparam.LCD_PRS = 1;                               //时钟2分频
    sparam.LCD_SEG0 = 0x0003;                         //LCD段使能
    LCD_Init(&sparam);                                //配置LCD模块
}
int main(void)
{
    uint32_t buf[6];                                  //显示缓存
    SystemClockConfig();                              //配置时钟
    User_GpioConfig();                                //配置IO
    User_LcdConfig();                                 //配置LCD
    buf[0] = 0x00000f0f;
    LCD_PixelWrite(LCD_Pixel_LCDD0,buf[0]);           //写像素寄存器
    LCD_Enable();                          //LCD驱动模块使能，LCD驱动开始显示
    /*演示LCD闪烁功能*/
```

```
    LCD_FlickerTimeConfig(2,1);                    //配置LCD闪烁时间
    LCD_FLIK_Enable();                             //LCD闪烁使能
    User_Delay();                                  //延时
    LCD_FLIK_Disable();                            //关LCD闪烁
    UserFunc();                                    //用户程序
}
```

第9章 通用异步收发器（UART）

9.1 功能概述

- 支持异步接收和异步发送
- 支持全/半双工通讯模式
- 支持 7/8/9 位数据传输格式
- 支持 4 级发送缓冲寄存器和 4 级接收缓冲寄存器
- 支持通讯传输波特率可配置
- 兼容 RS-232/RS-442/RS-485 的通讯接口
- 支持 PWM 调制输出，且 PWM 占空比线性可调
- 支持 UART 输入输出通讯端口极性可配置
- UART 接收端口支持红外唤醒功能

9.2 寄存器结构

UART 的寄存器定义于文件 ES8H296.h。芯片支持 4 个 UART，为 UART0/1/2/3。

```
typedef struct
{
 __IO UART_CON_Typedef CON;
 __IO UART_BRR_Typedef BRR;
 __O UART_TBW_Typedef TBW;
 __I UART_RBR_Typedef RBR;
 __I UART_TB01_Typedef TB01;
 __I UART_TB23_Typedef TB23;
 __I UART_RB01_Typedef RB01;
 __I UART_RB23_Typedef RB23;
 __IO UART_IE_Typedef IE;
 __IO UART_IF_Typedef IF;
} UART_TypeDef;
#define APB_BASE (0x40000000UL)
#define UART0_BASE (APB_BASE + 0x06000)
#define UART1_BASE (APB_BASE + 0x06400)
#define UART2_BASE (APB_BASE + 0x06800)
#define UART3_BASE (APB_BASE + 0x06C00)
#define UART0 ((UART_TypeDef *) UART0_BASE )
#define UART1 ((UART_TypeDef *) UART1_BASE )
#define UART2 ((UART_TypeDef *) UART2_BASE )
#define UART3 ((UART_TypeDef *) UART3_BASE )
```

9.3 宏定义

UART 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_uart.h 中。

```
/* 发送使能控制 */
#define UART0_TxEnable() (UART0->CON.TXEN = 1)
#define UART1_TxEnable() (UART1->CON.TXEN = 1)
#define UART2_TxEnable() (UART2->CON.TXEN = 1)
#define UART3_TxEnable() (UART3->CON.TXEN = 1)
#define UART0_TxDisable() (UART0->CON.TXEN = 0)
#define UART1_TxDisable() (UART1->CON.TXEN = 0)
#define UART2_TxDisable() (UART2->CON.TXEN = 0)
#define UART3_TxDisable() (UART3->CON.TXEN = 0)

/* 接收使能控制 */
#define UART0_RxEnable() (UART0->CON.RXEN = 1)
#define UART1_RxEnable() (UART1->CON.RXEN = 1)
#define UART2_RxEnable() (UART2->CON.RXEN = 1)
#define UART3_RxEnable() (UART3->CON.RXEN = 1)
#define UART0_RxDisable() (UART0->CON.RXEN = 0)
#define UART1_RxDisable() (UART1->CON.RXEN = 0)
#define UART2_RxDisable() (UART2->CON.RXEN = 0)
#define UART3_RxDisable() (UART3->CON.RXEN = 0)

/* 发送器复位 */
#define UART0_TxRst() (UART0->CON.TRST = 1)
#define UART1_TxRst() (UART1->CON.TRST = 1)
#define UART2_TxRst() (UART2->CON.TRST = 1)
#define UART3_TxRst() (UART3->CON.TRST = 1)

/* 接收器复位 */
#define UART0_RxRst() (UART0->CON.RRST = 1)
#define UART1_RxRst() (UART1->CON.RRST = 1)
#define UART2_RxRst() (UART2->CON.RRST = 1)
#define UART3_RxRst() (UART3->CON.RRST = 1)

#define UART_Tx3Pwm_Disable() (GPIOTXPC->TX3PEN = 0)
```

9.4 库函数

UART 库函数定义于 lib_uart.c 中，声明于 lib_uart.h 中。串口打印库函数定义于 lib_printf.c 中，声明于 lib_printf.h 中。

9. 4. 1 函数UART_Init

- 函数原型：void UART_Init(UART_TypeDef* UARTx, UART_InitStruType* UART_InitStruct)
- 功能描述：UART 初始化
- 输入参数：
  - ◇ UARTx：可以是 UART0/1/2/3
  - ◇ UART_InitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* UART初始化配置结构体定义 */
typedef struct
{
    UART_TYPE_TXFS   UART_StopBits;          //发送帧停止位选择
    UART_TYPE_DATAMOD   UART_TxMode;         //发送数据帧格式
    UART_TYPE_RTXP   UART_TxPolar;           //发送端口极性
    UART_TYPE_DATAMOD   UART_RxMode;         //接收数据帧格式
    UART_TYPE_RTXP   UART_RxPolar;           //接收端口极性
    uint32_t   UART_BuadRate;                //传输波特率
    UART_TYPE_BCS   UART_ClockSet;           //波特率发生器时钟选择
}UART_InitStruType;
```

发送帧停止位选择枚举类型 UART_TYPE_TXFS：

枚举元素	数值	描述
UART_StopBits_1	0x0	发送帧停止位：1 位
UART_StopBits_2	0x1	发送帧停止位：2 位

表 9-1 UART_TYPE_TXFS

数据格式枚举类型 UART_TYPE_DATAMOD：

枚举元素	数值	描述
UART_DataMode_7	0x0	7 位数据
UART_DataMode_8	0x1	8 位数据
UART_DataMode_9	0x2	9 位数据
UART_DataMode_7Odd	0x4	7 位数据+寄校验
UART_DataMode_7Even	0x5	7 位数据+偶校验
UART_DataMode_8Odd	0x6	8 位数据+寄校验
UART_DataMode_8Even	0x7	8 位数据+偶校验

表 9-2 UART_TYPE_DATAMOD

端口极性枚举类型 UART_TYPE_RTXP：

枚举元素	数值	描述
UART_Polar_Normal	0x0	发送端口极性：正常
UART_Polar_Opposite	0x1	发送端口极性：反向

表 9-3　UART_TYPE_RTXP

波特率发生器时钟选择枚举类型 UART_TYPE_BCS：

枚举元素	数值	描述
UART_Clock_1	0x1	波特率发生器时钟:PCLK
UART_Clock_2	0x2	波特率发生器时钟:PCLK/2
UART_Clock_3	0x3	波特率发生器时钟:PCLK/4

表 9-4　UART_TYPE_BCS

9. 4. 2　函数UART_ITConfig

- 函数原型：void　UART_ITConfig(UART_TypeDef*　UARTx,　UART_TYPE_IT UART_IT,　TYPE_FUNCEN　NewState)
- 功能描述：UART 中断配置
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - UART_IT：中断类型，详见表 9-5
  - NewState：使能/失能
- 返回值：无

中断选择枚举类型 UART_TYPE_IT：

枚举元素	数值	描述
UART_IT_TB	0x0001	发送缓冲器空中断
UART_IT_RB	0x0002	接收缓冲器满中断
UART_IT_RO	0x0004	接收数据溢出中断
UART_IT_FE	0x0008	接收帧错误中断
UART_IT_PE	0x0010	接收校验错误中断
UART_IT_TBWE	0x0020	发送数据错误中断
UART_IT_TXIDLE	0x1000	发送空闲标志中断
UART_IT_RXIDLE	0x2000	接收空闲标志中断

表 9-5　UART_TYPE_IT

9. 4. 3　函数UART_TBIMConfig

- 函数原型：void　UART_TBIMConfig(UART_TypeDef*　UARTx,

UART_TYPE_TRBIM Type)

- 功能描述：UART 发送缓冲器空中断模式选择
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - Type：空中断模式，详见表 9-6
- 返回值：无

发送、接收中断模式枚举类型 UART_TYPE_TRBIM：

枚举元素	数值	描述
UART_TBIM_Byte	0x0	中断：字节
UART_TBIM_HalfWord	0x1	中断：半字
UART_TBIM_Word	0x2	中断：字

表 9-6 UART_TYPE_TRBIM

9. 4. 4 函数UART_RBIMConfig

- 函数原型：void UART_RBIMConfig(UART_TypeDef* UARTx, UART_TYPE_TRBIM Type)
- 功能描述：UART 接收缓冲器满中断模式选择
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - Type：满中断模式，详见表 9-6
- 返回值：无

9. 4. 5 函数UART_Send

- 函数原型：
  - void UART_SendByte(UART_TypeDef* UARTx,uint8_t Temp08)
  - void UART_SendHalfWord(UART_TypeDef* UARTx,uint16_t Temp16)
  - void UART_SendWord(UART_TypeDef* UARTx,uint32_t Temp32)
- 功能描述：UART 发送字节、半字、字
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - Temp08/ Temp16/ Temp32：字节、半字、字数据
- 返回值：无

9. 4. 6 函数UART_Rec

- 函数原型：
  - uint8_t UART_RecByte(UART_TypeDef* UARTx)
  - uint16_t UART_RecHalfWord(UART_TypeDef* UARTx)
  - uint32_t UART_RecWord(UART_TypeDef* UARTx)
- 功能描述：UART 接收字节、半字、字
- 输入参数：可以是 UART0/1/2/3
- 返回值：读出的字节、半字、字数据

9. 4. 7 函数UART_TxxConfig

- 函数原型：
  - void UART_TX0Config(UART_TYPE_TXPLV Plv,UART_TYPE_TX0PS Ps)
  - void UART_TX1Config(UART_TYPE_TXPLV Plv,UART_TYPE_TX1PS Ps)
  - void UART_TX2Config(UART_TYPE_TXPLV Plv,UART_TYPE_TX2PS Ps)
  - void UART_TX3Config(UART_TYPE_TXPLV Plv,UART_TYPE_TX3PS Ps)
- 功能描述：TX0/1/2/3 脉宽调制输出配置
- 输入参数：
  - Plv：调制电平选择，详见表 9-7
  - Ps：调制 PWM 脉冲选择，详见表 9-8、表 9-9、表 9-10、表 9-11
- 返回值：无

脉宽调制电平枚举类型 UART_TYPE_TXPLV：

枚举元素	数值	描述
UART_TXPLV_Low	0x0	脉宽调制电平:低
UART_TXPLV_High	0x1	脉宽调制电平:高

表 9-7 UART_TYPE_TXPLV

脉宽 PWM 脉冲选择枚举类型 UART_TYPE_TX0PS：

枚举元素	数值	描述
UART_TX0PS_T16N0Out0	0x0	脉宽 PWM 脉冲选择:T16N0OUT0
UART_TX0PS_T16N0Out1	0x1	脉宽 PWM 脉冲选择:T16N0OUT1

表 9-8 UART_TYPE_TX0PS

脉宽 PWM 脉冲选择枚举类型 UART_TYPE_TX1PS：

枚举元素	数值	描述
UART_TX0PS_T16N1Out0	0x0	脉宽 PWM 脉冲选择:T16N1OUT0
UART_TX0PS_T16N1Out1	0x1	脉宽 PWM 脉冲选择:T16N1OUT1

表 9-9 UART_TYPE_TX1PS

脉宽 PWM 脉冲选择枚举类型 UART_TYPE_TX2PS：

枚举元素	数值	描述
UART_TX0PS_T16N2Out0	0x0	脉宽 PWM 脉冲选择:T16N2OUT0
UART_TX0PS_T16N2Out1	0x1	脉宽 PWM 脉冲选择:T16N2OUT1

表 9-10　UART_TYPE_TX2PS

脉宽 PWM 脉冲选择枚举类型 UART_TYPE_TX3PS：

枚举元素	数值	描述
UART_TX0PS_T16N3Out0	0x0	脉宽 PWM 脉冲选择:T16N3OUT0
UART_TX0PS_T16N3Out1	0x1	脉宽 PWM 脉冲选择:T16N3OUT1

表 9-11　UART_TYPE_TX3PS

9. 4. 8　函数UART_GetFlagStatus

- 函数原型：FlagStatus　UART_GetFlagStatus(UART_TypeDef*　UARTx, UART_TYPE_FLAG　UART_Flag)
- 功能描述：UART 获取标志位状态
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - UART_Flag：标志位选择，详见表 9-12
- 返回值：SET/RESET

标志位选择枚举类型 UART_TYPE_FLAG：

枚举元素	数值	描述
UART_FLAG_TB	0x0001	发送缓冲器空标志
UART_FLAG_RB	0x0002	接收缓冲器满标志
UART_FLAG_RO	0x0004	接收数据溢出标志
UART_FLAG_FE	0x0008	接收帧错误标志
UART_FLAG_PE	0x0010	接收校验错误标志
UART_FLAG_TBWE	0x0020	发送数据错误标志
UART_FLAG_TXIDLE	0x0100	发送空闲标志标志
UART_FLAG_RXIDLE	0x0200	接收空闲标志标志

表 9-12　UART_TYPE_FLAG

9. 4. 9　函数UART_GetITStatus

- 函数原型：ITStatus　UART_GetITStatus(UART_TypeDef*　UARTx,　UART_TYPE_IT UART_Flag)
- 功能描述：UART 获取中断状态,未使能相应中断时不会返回 SET

- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - UART_Flag：中断选择，详见表 9-5
- 返回值：SET/RESET

9. 4. 10 函数UART_ClearITPendingBit

- 函数原型：void UART_ClearITPendingBit(UART_TypeDef* UARTx, UART_CLR_IF UART_Flag)
- 功能描述：UART 标志位清除
- 输入参数：
  - UARTx：可以是 UART0/1/2/3
  - UART_Flag：标志位清除选择
- 返回值：无

标志位清除选择枚举类型 UART_CLR_IF：

枚举元素	数值	描述
UART_Clr_RO	0x0004	接收数据溢出标志
UART_Clr_FE	0x0008	接收帧错误标志
UART_Clr_PE	0x0010	接收校验错误标志
UART_Clr_TBWE	0x0020	发送数据错误标志

表 9-13 UART_CLR_IF

9. 4. 11 函数UART_printf

- 函数原型：ErrorStatus UART_printf(uint8_t* Data,...)
- 功能描述：串口格式化输出，使用方法同 C 库中的 printf 函数
- 输入参数：
  - Data：要发送到串口的内容的指针
  - ...：其他参数
- 返回值：成功、失败

该函数在 iDesigner 环境下支持的转义字符及格式字符(keil 环境下支持的转义字符及格式字符同 C 库中的 printf 函数)：

转义字符	描述	格式字符	描述
\r	回车	%d	带符号的十进制整数形式
\n	换行	%s	字符串

表 9-14 UART_printf 支持的格式

9. 4. 12 函数fputc

- 函数原型：int fputc(int ch, FILE* f)
- 功能描述：重定向 c 库中的函数 printf 到 UART（内部调用）
- 输入参数：（内部调用）
- 返回值：（内部调用）

9. 5 函数库应用示例

```
/* 初始化UART */
void User_UartInit(void)
{
    UART_InitStruType x;                                    //定义结构体
    x.UART_BuadRate = 57600;                                //波特率
    x.UART_ClockSet = UART_Clock_1;                         //时钟选择：Pclk
    x.UART_RxMode = UART_DataMode_8;                        //接收数据格式：8位数据
    x.UART_RxPolar = UART_Polar_Normal;                     //接收端口极性：正常
    x.UART_StopBits = UART_StopBits_1;                      //停止位：1
    x.UART_TxMode = UART_DataMode_8;                        //发送数据格式：8位数据
    x.UART_TxPolar = UART_Polar_Normal;                     //发送端口极性：正常
    UART_Init(UART0,&x);                                    //初始化UART0
    UART_RBIMConfig(UART0,UART_TBIM_Byte);                  //设置接收中断模式
    UART_ITConfig(UART0,UART_IT_RB,Enable);                 //使能接收中断
    NVIC_Init(NVIC_UART0_IRQn,NVIC_Priority_1,Enable); //设置优先级
}
/* printf使用示例 */
void Test_printfFunc(void)
{
    uint8_t buf = 0x55;
    printf("当前数据是(十六进制) 0x%x \r\n", buf);
    UART_printf("当前数据是(十进制) %d \r\n", buf);
}
```

第10章 增强型通用异步收发器（EUART）

10. 1 功能概述

- 配置为普通 UART 模式，与 UART 功能完全兼容
- 配置为 7816 模式，支持 7816 通讯协议
  - 支持异步接收器/发送器
  - 支持半双工通讯模式
  - 支持 8 位数据位和 1 位奇偶校验位数据传输格式
  - 支持自动重发重收功能
  - 支持可配置内部时钟输出
  - 支持双通道通讯可配置

10. 2 寄存器结构

EUART 的寄存器定义于文件 ES8H296.h。芯片支持 2 个 EUART，为 EUART0/EUART1。

```
typedef struct
{
 __IO EUART_CON_Typedef CON;
 __IO EUART_BRR_Typedef BRR;
 __O EUART_TBW_Typedef TBW;
 __I EUART_RBR_Typedef RBR;
 __I EUART_TB01_Typedef TB01;
 __I EUART_TB23_Typedef TB23;
 __I EUART_RB01_Typedef RB01;
 __I EUART_RB23_Typedef RB23;
 __IO EUART_IE_Typedef IE;
 __IO EUART_IF_Typedef IF;
 uint32_t RESERVED0[2] ;
 __IO EUART_MOD_Typedef MOD;
} EUART_TypeDef;
#define APB_BASE (0x40000000UL)
#define EUART0_BASE (APB_BASE + 0x07000)
#define EUART0 ((EUART_TypeDef *) EUART0_BASE )
#define EUART1_BASE (APB_BASE + 0x07400)
#define EUART1 ((EUART_TypeDef *) EUART1_BASE )
```

10. 3 宏定义

EUART 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_euart.h 中。

```
/* 发送使能控制 */
#define EUART0_TxEnable()   (EUART0->CON.TXEN = 1)
#define EUART1_TxEnable()   (EUART1->CON.TXEN = 1)
#define EUART0_TxDisable() (EUART0->CON.TXEN = 0)
#define EUART1_TxDisable() (EUART1->CON.TXEN = 0)
/* 接收使能控制 */
#define EUART0_RxEnable()   (EUART0->CON.RXEN = 1)
#define EUART1_RxEnable()   (EUART1->CON.RXEN = 1)
#define EUART0_RxDisable() (EUART0->CON.RXEN = 0)
#define EUART1_RxDisable() (EUART1->CON.RXEN = 0)
/* 发送器复位 */
#define EUART0_TxRst() (EUART0->CON.TRST = 1)
#define EUART1_TxRst() (EUART1->CON.TRST = 1)
/* 接收器复位 */
#define EUART0_RxRst() (EUART0->CON.RRST = 1)
#define EUART1_RxRst() (EUART1->CON.RRST = 1)
/* U7816 软件复位 */
#define EUART0_U7816_REST() ( EUART0->MOD.ERST = 1)
#define EUART1_U7816_REST() ( EUART1->MOD.ERST = 1)
```

10. 4 库函数

EUART 库函数定义于 lib_euart.c 中，声明于 lib_euart.h 中。

10. 4. 1 函数EUART_ModeConfig

- 函数原型：void EUART_ModeConfig(EUART_TypeDef* EUARTx,
  EUART_TYPE_MODE Mode)
- 功能描述：EUART 工作模式选择
- 输入参数：
  - EUARTx：可以是 EUART0
  - Mode：工作模式，详见表 10-1
- 返回值：无

EUART 工作模式选择枚举类型 EUART_TYPE_MODE：

枚举元素	数值	描述
EUART_Mode_U7816	0x1	7816 模式
EUART_Mode_Uart	0x0	普通 uart 模式

表 10-1 EUART_TYPE_MODE

10. 4. 2 函数EUART_Init

- 函数原型：void EUART_Init(EUART_TypeDef* EUARTx, EUART_InitStruType* EUART_InitStruct)
- 功能描述：EUART 初始化
- 输入参数：
  - ◇ EUARTx：可以是 EUART0
  - ◇ EUART_InitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* EUART初始化配置结构体定义 */
typedef struct
{
    EUART_TYPE_TXFS   EUART_StopBits;        //发送帧停止位选择
    EUART_TYPE_DATAMOD   EUART_TxMode;       //发送数据帧格式
    EUART_TYPE_RTXP   EUART_TxPolar;         //发送端口极性
    EUART_TYPE_DATAMOD   EUART_RxMode;       //接收数据帧格式
    EUART_TYPE_RTXP   EUART_RxPolar;         //接收端口极性
    uint32_t   EUART_BuadRate;               //传输波特率
    EUART_TYPE_BCS   EUART_ClockSet;         //波特率发生器时钟选择
}EUART_InitStruType;
```

发送帧停止位选择枚举类型 EUART_TYPE_TXFS：

枚举元素	数值	描述
EUART_StopBits_1	0x0	发送帧停止位：1 位
EUART_StopBits_2	0x1	发送帧停止位：2 位

表 10-2 EUART_TYPE_TXFS

数据格式枚举类型 EUART_TYPE_DATAMOD：

枚举元素	数值	描述
EUART_DataMode_7	0x0	7 位数据
EUART_DataMode_8	0x1	8 位数据
EUART_DataMode_9	0x2	9 位数据
EUART_DataMode_7Odd	0x4	7 位数据+奇校验
EUART_DataMode_7Even	0x5	7 位数据+偶校验
EUART_DataMode_8Odd	0x6	8 位数据+奇校验
EUART_DataMode_8Even	0x7	8 位数据+偶校验

表 10-3 EUART_TYPE_DATAMOD

端口极性枚举类型 EUART_TYPE_RTXP：

枚举元素	数值	描述
EUART_Polar_Normal	0x0	发送端口极性：正常
EUART_Polar_Opposite	0x1	发送端口极性：反向

表 10-4　EUART_TYPE_RTXP

波特率发生器时钟选择枚举类型 EUART_TYPE_BCS：

枚举元素	数值	描述
EUART_Clock_1	0x1	波特率发生器时钟:PCLK
EUART_Clock_2	0x2	波特率发生器时钟:PCLK/2
EUART_Clock_3	0x3	波特率发生器时钟:PCLK/4

表 10-5　EUART_TYPE_BCS

10. 4. 3　函数EUART_BaudConfig

- 函数原型：void　EUART_BaudConfig(EUART_TypeDef*　EUARTx,　uint32_t BaudRate,　EUART_TYPE_BCS　ClockSet)
- 功能描述：EUART 波特率设置
- 输入参数：
  - EUARTx：可以是 EUART0
  - BaudRate：目标波特率（单位 HZ）
  - ClockSet：波特率发生器时钟选择，详见表 10-5
- 返回值：无

10. 4. 4　函数EUART_ITConfig

- 函数原型：void　EUART_ITConfig(EUART_TypeDef*　EUARTx,　EUART_TYPE_IT EUART_IT,　TYPE_FUNCEN　NewState)
- 功能描述：EUART 中断配置
- 输入参数：
  - EUARTx：可以是 EUART0
  - EUART_IT：中断类型，详见表 10-6
  - NewState：使能/失能
- 返回值：无

中断类型枚举类型 EUART_TYPE_IT：

枚举元素	数值	描述
EUART_IT_TB	0x0001	发送缓冲器空中断
EUART_IT_RB	0x0002	接收缓冲器满中断

枚举元素	数值	描述
EUART_IT_RO	0x0004	接收数据溢出中断
EUART_IT_FE	0x0008	接收帧错误中断
EUART_IT_PE	0x0010	接收校验错误中断
EUART_IT_TBWE	0x0020	发送数据错误中断
EUART_IT_ARTE	0x0040	自动重发失败中断
EUART_IT_RNA	0x0080	接收 NACK 中断
EUART_IT_TXIDLE	0x1000	发送空闲标志中断
EUART_IT_RXIDLE	0x2000	接收空闲标志中断

表 10-6　EUART_TYPE_IT

10. 4. 5　函数EUART_TBIMConfig

- 函数原型：void　EUART_TBIMConfig(EUART_TypeDef*　EUARTx, EUART_TYPE_TRBIM　Type)
- 功能描述：EUART 发送缓冲器空中断模式选择
- 输入参数：
  - EUARTx：可以是 EUART0
  - Type：空中断模式,详见表 10-7
- 返回值：无

发送、接收中断模式枚举类型 EUART_TYPE_TRBIM：

枚举元素	数值	描述
EUART_TBIM_Byte	0x0	中断：字节
EUART_TBIM_HalfWord	0x1	中断：半字
EUART_TBIM_Word	0x2	中断：字

表 10-7　EUART_TYPE_TRBIM

10. 4. 6　函数EUART_RBIMConfig

- 函数原型：void　EUART_RBIMConfig(EUART_TypeDef*　EUARTx, EUART_TYPE_TRBIM　Type)
- 功能描述：EUART 接收缓冲器满中断模式选择
- 输入参数：
  - EUARTx：可以是 EUART0
  - Type：满中断模式，详见表 10-7
- 返回值：无

10. 4. 7 函数EUART_GetFlagStatus

- 函数原型：FlagStatus EUART_GetFlagStatus(EUART_TypeDef* EUARTx, EUART_TYPE_FLAG EUART_Flag)
- 功能描述：EUART 获取标志位状态
- 输入参数：
  - EUARTx：可以是 EUART0
  - EUART_Flag：标志位选择
- 返回值：SET/RESET

标志位选择枚举类型 EUART_TYPE_FLAG：

枚举元素	数值	描述
EUART_FLAG_TB	0x0001	发送缓冲器空标志
EUART_FLAG_RB	0x0002	接收缓冲器满标志
EUART_FLAG_RO	0x0004	接收数据溢出标志
EUART_FLAG_FE	0x0008	接收帧错误标志
EUART_FLAG_PE	0x0010	接收校验错误标志
EUART_FLAG_TBWE	0x0020	发送数据错误标志
EUART_FLAG_ARTE	0x0040	自动重发失败标志
EUART_FLAG_RNA	0x0080	接收 NACK 标志
EUART_FLAG_TXIDLE	0x0100	发送空闲标志标志
EUART_FLAG_RXIDLE	0x0200	接收空闲标志标志

表 10-8 EUART_TYPE_FLAG

10. 4. 8 函数EUART_GetITStatus

- 函数原型：ITStatus EUART_GetITStatus(EUART_TypeDef* EUARTx, EUART_TYPE_IT EUART_Flag)
- 功能描述：UART 获取中断状态,未使能相应中断时不会返回 SET
- 输入参数：
  - EUARTx：可以是 EUART0
  - EUART_Flag：中断选择，详见表 10-6
- 返回值：SET/RESET

10. 4. 9 函数EUART_ClearITPendingBit

- 函数原型：void EUART_ClearITPendingBit(EUART_TypeDef* EUARTx, EUART_CLR_IF EUART_Flag)
- 功能描述：EUART 标志位清除

- 输入参数：
  - EUARTx：可以是 EUART0
  - EUART_Flag：标志位清除中断选择
- 返回值：无

标志位清除中断选择枚举类型 EUART_CLR_IF：

枚举元素	数值	描述
EUART_Clr_RO	0x0004	接收数据溢出标志
EUART_Clr_FE	0x0008	接收帧错误标志
EUART_Clr_PE	0x0010	接收校验错误标志
EUART_Clr_TBWE	0x0020	发送数据错误标志
EUART_Clr_ARTE	0x0040	自动重发失败标志
EUART_Clr_RNA	0x0080	接收 NACK 标志

表 10-9　EUART_CLR_IF

10. 4. 10 函数U7816_Init

- 函数原型：void　U7816_Init(EUART_TypeDef*　EUARTx,　U7816_InitStruType* U7816_InitStruct)
- 功能描述：
- 输入参数：
  - EUARTx：可以是 EUART0
  - U7816_InitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* U7816模块初始化配置结构体定义 */
typedef struct
{
    EUART_TYPE_BCS   U7816_ClockSet;          //波特率发生器时钟选择
    uint32_t   U7816_BaudRate;                //传输波特率
    TYPE_FUNCEN   U7816_ECK0;                 //7816ECK0使能位
    TYPE_FUNCEN   U7816_ECK1;                 //7816ECK1使能位
    U7816_TYPE_CHS   U7816_EIOCh;             //EIO通讯通道选择位
    U7816_TYPE_EIOC   U7816_EIODir;           //EIO端口方向控制
    U7816_TYPE_DAS   U7816_DataForm;          //数据格式选择位
    U7816_TYPE_PS   U7816_DataVerify;         //数据奇偶校验位
    TYPE_FUNCEN   U7816_AutoRetryTX;          //自动重发使能位
    TYPE_FUNCEN   U7816_AutoRetryRX;          //自动重收使能位
    U7816_TYPE_TNAS   U7816_NACK_Width;       //NACK信号宽度
    U7816_TYPE_ARTS   U7816_RetryTimes;       //自动重发次数
```

```
    U7816_TYPE_CKS   U7816_CLKS;                    //时钟源选择
    uint8_t   U7816_ETUTime;                        //ETU保护时间选择
}U7816_InitStruType;
```

EIO 通讯通道选择枚举类型 U7816_TYPE_CHS：

枚举元素	数值	描述
U7816_CHS_EIO0	0x0	7816 IO 端口 0
U7816_CHS_EIO1	0x1	7816 IO 端口 1

表 10-10　U7816_TYPE_CHS

EIO 端口方向选择枚举类型 U7816_TYPE_EIOC：

枚举元素	数值	描述
U7816_EIODir_In	0x0	EIO 接收数据
U7816_EIODir_Out	0x1	EIO 发送数据

表 10-11　U7816_TYPE_EIOC

数据格式选择枚举类型 U7816_TYPE_DAS：

枚举元素	数值	描述
U7816_DataForm_Normal	0x0	数据格式为正向
U7816_DataForm_Contrary	0x1	数据格式为反向

表 10-12　U7816_TYPE_DAS

数据奇偶校验选择枚举类型 U7816_TYPE_PS：

枚举元素	数值	描述
U7816_Verify_Odd	0x0	奇数校验
U7816_Verify_Even	0x1	偶数校验

表 10-13　U7816_TYPE_PS

NACK 信号宽度选择枚举类型 U7816_TYPE_TNAS：

枚举元素	数值	描述
U7816_NACKWidth_1ETU	0x0	NACK 信号宽度：1 个 ETU
U7816_NACKWidth_1P5ETU	0x1	NACK 信号宽度：1.5 个 ETU
U7816_NACKWidth_2ETU	0x2	NACK 信号宽度：2 个 ETU

表 10-14　U7816_TYPE_TNAS

自动重发次数选择枚举类型 U7816_TYPE_ARTS：

枚举元素	数值	描述
U7816_RetryTimes_1	0x0	自动重发次数：1 次

枚举元素	数值	描述
U7816_RetryTimes_2	0x1	自动重发次数：2 次
U7816_RetryTimes_3	0x2	自动重发次数：3 次
U7816_RetryTimes_N	0x3	自动重发次数：无限次，直到发送成功

表 10-15　U7816_TYPE_ARTS

时钟源选择枚举类型 U7816_TYPE_CKS：

枚举元素	数值	描述
U7816_PCLK_1	0x0	PCLK 时钟
U7816_PCLK_2	0x1	PCLK 时钟 2 分频
U7816_PCLK_4	0x2	PCLK 时钟 4 分频
U7816_PCLK_8	0x3	PCLK 时钟 8 分频

表 10-16　U7816_TYPE_CKS

10. 4. 11 函数EUART_EIOChConfig

- 函数原型：void　EUART_EIOChConfig (EUART_TypeDef*　EUARTx,
  U7816_TYPE_CHS　U7816_IO)
- 功能描述：U7816 EIO 通讯通道选择
- 输入参数：
  - EUARTx：可以是 EUART0
  - U7816_IO：IO 端口号，详见表 10-10
- 返回值：无

10. 4. 12 函数EUART_EIODirection

- 函数原型：void　EUART_EIODirection (EUART_TypeDef*
  EUARTx,U7816_TYPE_EIOC
  Dir)
- 功能描述：U7816 EIO 端口方向控制
- 输入参数：
  - EUARTx：可以是 EUART0
  - Dir：IO 端口状态，详见表 10-11
- 返回值：无

10. 4. 13 函数EUART_Send

- 函数原型：
  - void　EUART_SendByte (EUART_TypeDef*　EUARTx, uint8_t　ByteData)

- ◇ void EUART_SendHalfWord (EUART_TypeDef* EUARTx, uint16_t HalfWordData)
- ◇ void EUART_SendWord (EUART_TypeDef* EUARTx, uint32_t WordData)

◆ 功能描述：U7816 写字节、半字、字数据

◆ 输入参数：

- ◇ EUARTx：可以是 EUART0
- ◇ ByteData/ HalfWordData/ WordData：要发送的字节、半字、字数据

◆ 返回值：无

10. 4. 14 函数EUART_Rec

◆ 函数原型：

- ◇ uint8_t EUART_RecByte (EUART_TypeDef* EUARTx)
- ◇ uint16_t EUART_RecHalfWord (EUART_TypeDef* EUARTx)
- ◇ uint32_t EUART_RecWord (EUART_TypeDef* EUARTx)

◆ 功能描述：U7816 读字节、半字、字数据

◆ 输入参数：EUARTx：可以是 EUART0

◆ 返回值：读到的字节、半字、字数据

10. 5 函数库应用示例

```
/* 配置EUART为7816模式 */
void User_EuartConfig(void)
{
    U7816_InitStruType sparam;                          //定义结构
    EUART_ModeConfig(EUART0,EUART_Mode_U7816);          //7816模式
    sparam. U7816_ClockSet = EUART_Clock_2;             //时钟
    sparam. U7816_BaudRate = 6720;                      //配置波特率
    sparam.U7816_ECK0  = Disable;                       //ECK0不使能
    sparam.U7816_ECK1  =  Enable;                       //ECK1使能
    sparam.U7816_EIOCh = U7816_CHS_EIO1;                //IO端口1
    sparam.U7816_EIODir = U7816_EIODir_In;              //EIO接收数据
    sparam.U7816_DataForm = U7816_DataForm_Normal;      //数据格式为正向
    sparam.U7816_DataVerify = U7816_Verify_Even;        //数据偶数校验
    sparam.U7816_AutoRetryTX = Enable;                  //自动重发使能
    sparam.U7816_AutoRetryRX = Enable;                  //自动重收使能
    sparam.U7816_NACK_Width = U7816_NACKWidth_1ETU;     //NACK信号宽度
    sparam.U7816_RetryTimes = U7816_RetryTimes_3;       //自动重发3次
    sparam.U7816_CLKS = U7816_PCLK_8;                   //PCLK时钟8分频
    sparam.U7816_ETUTime = 1;                           //3个ETU时间
    U7816_Init(EUART0, &sparam);                        //7816初始化
}
```

第11章 IIC串行总线

11. 1 功能概述

- 支持单主控模式
  - 支持自动重复寻呼功能
  - 支持自动发送“停止位”功能
  - 支持数据应答延迟功能
  - 支持数据帧传输间隔功能
  - 支持软件触发“起始位”
  - 支持软件触发“停止位”
  - 支持软件触发数据接收，接收模式可配
- 支持从动模式
  - 支持 7 位从机地址可配
  - 支持从机地址匹配中断标志
  - 支持接收“停止位”中断标志
  - 支持时钟线自动下拉等待请求功能
  - 支持自动发送“未应答”功能
- 支持 4 级发送缓冲器和 4 级接收缓冲器
- 通讯端口 SCL 和 SDA，均支持输出模式可配置
- 通讯端口 SCL 和 SDA 支持 16 倍速采样器可配置
- 支持发送和接收缓冲器空/满中断
- 支持起始位中断、停止位中断
- 支持接收数据溢出中断、发送数据写错误中断

11. 2 寄存器结构

IIC 的寄存器定义于文件 ES8H296.h。芯片支持 1 个 IIC。

```
typedef struct
{
    /* Offset: 0x000 (R/W)  控制寄存器  */
__IO I2C_CON_Typedef CON; ;
    /* Offset: 0x004 (R/W)  工作模式寄存器  */
__IO I2C_MOD_Typedef MOD ;
    /* Offset: 0x008 (R/W)  中断使能寄存器  */
```

```
__IO I2C_IE_Typedef IE ;
    /* Offset: 0x00C (R/W)  中断状态寄存器  */
__IO I2C_IF_Typedef IF;
    /* Offset: 0x010 ( /W)  发送数据写入寄存器  */
__O I2C_TBW_Typedef TBW ;
    /* Offset: 0x014 (R/ )  接收数据读取寄存器  */
__I I2C_RBR_Typedef RBR;
    /* Offset: 0x018 (R/ )  发送缓冲寄存器  */
__I I2C_TB_Typedef TB ;
    /* Offset: 0x01C (R/ )  接收缓冲寄存器  */
__I I2C_RB_Typedef RB ;
    /* Offset: 0x020 (R/ )  状态寄存器  */
__I I2C_STA_Typedef STA;
} I2C_TypeDef;
#define APB_BASE (0x40000000UL)
#define I2C_BASE (APB_BASE + 0x09000)
#define I2C ((I2C_TypeDef *) I2C_BASE )
```

11. 3 宏定义

IIC 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_iic.h 中。

```
/* IIC模块使能控制  */
#define I2C_Enable() (I2C->CON.EN = 1)
#define I2C_Disable() (I2C->CON.EN = 0)
/* IIC模块复位  */
#define I2C_Reset() (I2C->CON.RST = 1)
/* IIC时基使能控制  */
#define I2C_TJEnable() (I2C->CON.TJE = 1)
#define I2C_TJDisable() (I2C->CON.TJE = 0)
/* IIC主控模式读写控制  */
#define I2C_Read() (I2C->CON.RW = 1)
#define I2C_Write() (I2C->CON.RW = 0)
/* IIC时钟自动下拉等待使能控制（仅从机模式支持）  */
#define I2C_CSEnable() (I2C->MOD.CSE = 1)
#define I2C_CSDisable() (I2C->MOD.CSE = 0)
/* IIC自动发送未应答使能控制（仅从机模式支持）  */
#define I2C_ANAEnable() (I2C->MOD.ANAE = 1)
#define I2C_ANADisable() (I2C->MOD.ANAE = 0)
/* IIC自动寻呼使能控制（仅主机模式支持）  */
#define I2C_SRAEnable() (I2C->MOD.SRAE = 1)
#define I2C_SRADisable() (I2C->MOD.SRAE = 0)
/* IIC自动结束使能控制（仅主机模式支持）  */
#define I2C_SPAEnable() (I2C->MOD.SPAE = 1)
#define I2C_SPADisable() (I2C->MOD.SPAE = 0)
```

```
/* IIC起始位触发（仅主机模式支持） */
#define I2C_SRTrigger() (I2C->MOD.SRT=1)
/* IIC停止位触发（仅主机模式支持） */
#define I2C_SPTrigger() (I2C->MOD.SPT = 1)
/* IIC接收数据触发（仅主机模式支持） */
#define I2C_RDTrigger() (I2C->MOD.RDT = 1)
/* IIC总线释放 */
#define I2C_Release() (I2C->MOD.BLD = 1)
/* IIC发送应答设置（仅从机模式支持） */
#define I2C_TACK() (I2C->MOD.TAS = 1)
#define I2C_TNACK() (I2C->MOD.TAS = 0)
```

11. 4 库函数

IIC 库函数定义于 lib_iic.c 中，声明于 lib_iic.h 中。

11. 4. 1 函数IIC_Init

- 函数原型：void I2C_Init(I2C_InitStruType* I2C_InitStruct)
- 功能描述：IIC 初始化
- 输入参数：
  - ◇ I2C_InitStruct：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* IIC初始化配置结构体定义 */
typedef struct
{
    I2C_TYPE_PINOD   I2C_SclOd;          //SCL端口输出模式
    I2C_TYPE_PINOD   I2C_SdaOd;          //SDA端口输出模式
    TYPE_FUNCEN   I2C_16XSamp;           //端口16倍速采样使能
    uint32_t   I2C_Clk;                  //I2C频率
    I2C_TYPE_MODE   I2C_Mode;            //工作模式
    TYPE_FUNCEN   I2C_AutoStop;          //自动停止
    TYPE_FUNCEN   I2C_AutoCall;          //自动寻呼
}I2C_InitStruType;
```

引脚开漏设置枚举类型 I2C_TYPE_PINOD：

枚举元素	数值	描述
I2C_PinMode_PP	0x0	端口输出模式：推挽
I2C_PinMode_OD	0x1	端口输出模式：开漏

表 11-1 I2C_TYPE_PINOD

工作模式选择枚举类型 I2C_TYPE_MODE：

枚举元素	数值	描述
I2C_Mode_Master	0x0	工作模式：主
I2C_Mode_Slave	0x1	工作模式：从

表 11-2　I2C_TYPE_MODE

11. 4. 2　函数I2C_ITConfig

- 函数原型：void I2C_ITConfig(I2C_TYPE_IT I2C_IT,TYPE_FUNCEN NewState)
- 功能描述：I2C 中断配置
- 输入参数：
  - ◇ I2C_IT：需要配置的中断，详见表 11-3
  - ◇ NewState：使能或关闭
- 返回值：无

中断选择枚举类型 I2C_TYPE_IT：

枚举元素	数值	描述
I2C_IT_SR	0x0001	起始位中断
I2C_IT_SP	0x0002	停止位中断
I2C_IT_TB	0x0004	发送缓冲空中断
I2C_IT_RB	0x0008	接收缓冲满中断
I2C_IT_TE	0x0010	发送数据错误中断
I2C_IT_RO	0x0020	接收数据溢出中断
I2C_IT_NA	0x0040	未应答 NACK 中断
I2C_IT_TBWE	0x0080	发送数据写错误中断
I2C_IT_IDLE	0x1000	空闲中断

表 11-3　I2C_TYPE_IT

11. 4. 3　函数I2C_SendAddress

- 函数原型：void I2C_SendAddress(uint8_t I2C_Address,I2C_TYPE_RWMODE Mode)
- 功能描述：I2C 发送从机地址（主控模式时使用）
- 输入参数：
  - ◇ I2C_Address：7 位从机地址，左对齐（0x00~0xFE）
  - ◇ Mode：读或写,详见表 11-4
- 返回值：无

读写模式选择枚举类型 I2C_TYPE_RWMODE：

枚举元素	数值	描述
I2C_Mode_Write	0x0	读模式
I2C_Mode_Read	0x1	写模式

表 11-4　I2C_TYPE_RWMODE

11. 4. 4　函数I2C_SetAddress

- 函数原型：void I2C_SetAddress(uint8_t I2C_Address)
- 功能描述：I2C 设置地址（适用于从机模式）
- 输入参数：
  - ◇ I2C_Address：7 位从机地址，左对齐（0x00~0xFE）
- 返回值：无

11. 4. 5　函数I2C_RecModeConfig

- 函数原型：void I2C_RecModeConfig(I2C_TYPE_RECMODE RecType)
- 功能描述：I2C 配置接收模式
- 输入参数：
  - ◇ RecType：接收模式，详见表 11-5
- 返回值：无

接收模式选择枚举类型 I2C_TYPE_RECMODE：

枚举元素	数值	描述
I2C_RecMode_0	0x0	接收 1 字节，发送 ACK
I2C_RecMode_1	0x1	接收 1 字节，发送 NACK
I2C_RecMode_2	0x2	接收 2 字节，每字节发送 ACK
I2C_RecMode_3	0x3	接收 2 字节，前一字节发送 ACK，后一字节发送 NACK
I2C_RecMode_4	0x4	接收 4 字节，每字节发送 ACK
I2C_RecMode_5	0x5	接收 4 字节，前 3 字节发送 ACK，后一字节发送 NACK
I2C_RecMode_6	0x6	连续接收，每个字节发送 ACK
I2C_RecMode_7	0x7	完成该字节接收，发送 NACK

表 11-5　I2C_TYPE_RECMODE

11. 4. 6　函数 I2C_TBIMConfig

- 函数原型：void　I2C_TBIMConfig(I2C_TYPE_TRBIM Type)
- 功能描述：I2C 发送缓冲器空中断模式选择

- 输入参数：
  - Type：空中断模式，详见表 11-6
- 返回值：无

发送、接收中断模式选择枚举类型 I2C_TYPE_TRBIM：

枚举元素	数值	描述
I2C_TBIM_Byte	0x0	中断：字节空
I2C_TBIM_HalfWord	0x1	中断：半字空
I2C_TBIM_Word	0x2	中断：字空

表 11-6 I2C_TYPE_TRBIM

11.4.7 函数I2C_RBIMConfig

- 函数原型：void I2C_RBIMConfig(I2C_TYPE_TRBIM Type)
- 功能描述：I2C 接收缓冲器满中断模式选择
- 输入参数：
  - Type：满中断模式，详见表 11-6
- 返回值：无

11.4.8 函数I2C_AckDelay

- 函数原型：void I2C_AckDelay(I2C_TYPE_ADLY Type,TYPE_FUNCEN NewStatus)
- 功能描述：I2C 应答延时配置
- 输入参数：
  - Type：延时时间，详见表 11-7
  - NewStatus：使能、失能
- 返回值：无

应答延时选择枚举类型 I2C_TYPE_ADLY：

枚举元素	数值	描述
IIC_AckDelay_0P5	0x0	应答延时：0.5 个时钟周期
IIC_AckDelay_1	0x1	应答延时：1 个时钟周期
IIC_AckDelay_1P5	0x2	应答延时：1.5 个时钟周期
IIC_AckDelay_2	0x3	应答延时：2 个时钟周期
IIC_AckDelay_2P5	0x4	应答延时：2.5 个时钟周期
IIC_AckDelay_3	0x5	应答延时：3 个时钟周期
IIC_AckDelay_3P5	0x6	应答延时：3.5 个时钟周期
IIC_AckDelay_4	0x7	应答延时：4 个时钟周期

表 11-7 I2C_TYPE_ADLY

11. 4. 9 函数I2C_TISConfig

- 函数原型：void I2C_TISConfig(I2C_TYPE_TIS Time)
- 功能描述：I2C 数据帧传输间隔设置
- 输入参数：
  - Time：传输间隔，详见表 11-8
- 返回值：无

数据帧传输间隔选择枚举类型 I2C_TYPE_TIS：

枚举元素	数值	描述
I2C_TI_Disable	0x0	传输间隔：0
I2C_TI_1	0x1	传输间隔：1
I2C_TI_2	0x2	传输间隔：2
I2C_TI_3	0x3	传输间隔：3
I2C_TI_4	0x4	传输间隔：4
I2C_TI_5	0x5	传输间隔：5
I2C_TI_6	0x6	传输间隔：6
I2C_TI_7	0x7	传输间隔：7
I2C_TI_8	0x8	传输间隔：8
I2C_TI_9	0x9	传输间隔：9
I2C_TI_10	0XA	传输间隔：10
I2C_TI_11	0XB	传输间隔：11
I2C_TI_12	0XC	传输间隔：12
I2C_TI_13	0xD	传输间隔：13
I2C_TI_14	0xE	传输间隔：14
I2C_TI_15	0xF	传输间隔：15

表 11-8 I2C_TYPE_TIS

11. 4. 10 函数I2C_Send

- 函数原型：
  - Void I2C_SendByte(uint8_t Byte)
  - void I2C_SendHalfWord(uint16_t HalfWord)
  - void I2C_SendWord(uint32_t Word)
- 功能描述：I2C 发送字节、半字、字数据
- 输入参数：
  - Byte/HalfWord/Word：字节、半字、字数据
- 返回值：无

11. 4. 11 函数I2C_Rec

- 函数原型：
  - uint8_t I2C_RecByte(void)
  - uint16_t I2C_RecHalfWord(void)
  - uint32_t I2C_RecWord(void)
- 功能描述：I2C 接收字节、半字、字数据
- 输入参数：无
- 返回值：接收的字节、半字、字数据

11. 4. 12 函数I2C_GetRWMode

- 函数原型：I2C_TYPE_RWMODE I2C_GetRWMode(void)
- 功能描述：I2C 工作读写状态读取
- 输入参数：无
- 返回值：读或写，详见表 11-4

11. 4. 13 函数I2C_GetTBStatus

- 函数原型：FlagStatus I2C_GetTBStatus(void)
- 功能描述：获取发送缓冲寄存器状态,TB0-TB3 全空则返回 SET,否则返回 RESET
- 输入参数：无
- 返回值：SET/RESET

11. 4. 14 函数I2C_GetFlagStatus

- 函数原型：FlagStatus I2C_GetFlagStatus(I2C_TYPE_FLAG I2C_Flag)
- 功能描述：I2C 获取标志位状态
- 输入参数：
  - I2C_Flag：标志位选择，详见表 11-9
- 返回值：SET/RESET

标志位选择枚举类型 I2C_TYPE_FLAG：

枚举元素	数值	描述
IIC_Flag_SR	0x0001	起始位中断标志
IIC_Flag_SP	0x0002	停止位中断标志
IIC_Flag_TB	0x0004	发送缓冲空中断标志

枚举元素	数值	描述
IIC_Flag_RB	0x0008	接收缓冲满中断标志
IIC_Flag_TE	0x0010	发送数据错误中断标志
IIC_Flag_RO	0x0020	接收数据溢出中断标志
IIC_Flag_NA	0x0040	未应答 NACK 中断标志
IIC_Flag_TBWE	0x0080	发送数据写错误中断标志
I2C_Flag_TIDLE	0x0100	I2C 发送空闲中断标志位

表 11-9 I2C_TYPE_FLAG

11. 4. 15 函数I2C_GetITStatus

- 函数原型：FlagStatus I2C_GetITStatus(I2C_TYPE_IT I2C_Flag)
- 功能描述：I2C 获取中断状态,未使能相应中断时不会返回 SET
- 输入参数：
  - ◇ I2C_Flag：选择需要的中断，详见表 11-3
- 返回值：SET/RESET

11. 4. 16 函数I2C_ClearITPendingBit

- 函数原型：void I2C_ClearITPendingBit(I2C_CLR_IF I2C_IT)
- 功能描述：I2C 中断标志清除
- 输入参数：
  - ◇ I2C_IT：选择要清除的标志位，详见表 11-10
- 返回值：无

标志位清除选择枚举类型 I2C_CLR_IF：

枚举元素	数值	描述
I2C_Clr_SR	0x0001	起始位中断标志
I2C_Clr_SP	0x0002	停止位中断标志
I2C_Clr_TE	0x0010	发送数据错误中断标志
I2C_Clr_RO	0x0020	接收数据溢出中断标志
I2C_Clr_NA	0x0040	未应答 NACK 中断标志
I2C_Clr_TBWE	0x0080	发送数据写错误中断标志
I2C_Clr_TIDLE	0x1000	I2C 发送空闲

表 11-10 I2C_CLR_IF

11. 5 函数库应用示例

```
/* I2C1初始化程序 */
void User_I2CInit(void)
{
    I2C_InitStruType y;                                    //定义结构
    y.I2C_16XSamp = Disable;                               //16倍采样设置
    y.I2C_AutoStop = Enable;                               //自动停止
    y.I2C_Clk = 400000;                                    //通讯频率400K
    y.I2C_Mode = I2C_Mode_Master;                          //主从设置
    y.I2C_SclOd = I2C_PinMode_OD;                          //端口开漏
    y.I2C_SdaOd = I2C_PinMode_OD;                          //端口开漏
    I2C_Init(&y);                                          //初始化IIC1
    NVIC_Init(NVIC_I2C_IRQn,NVIC_Priority_1,Enable);       //NVIC设置
    I2C_ITConfig(IIC_IT_SR,Enable);                        //使能RB中断
    I2C_Enable;                                            //使能IIC1
}
```

第12章 SPI串行总线

12. 1 功能概述

- 支持主控模式、从动模式
- 支持 4 种数据传输格式
- 支持主控模式通讯时钟速率可配置
- 支持 4 级发送缓冲器和 4 级接收缓冲器
- 支持发送和接收缓冲器空/满中断
- 支持接收数据溢出中断、发送数据写错误中断、从动模式的发送数据错误中断
- 支持从动模式的片选变化中断、主控模式的空闲状态中断
- 支持主控模式延迟接收
- 支持主控模式发送间隔

12. 2 寄存器结构

SPI 的寄存器定义于文件 ES8H296.h。芯片支持 1 个 SPI。

```
typedef struct
{
 __IO SPI_CON_Typedef CON;
 uint32_t RESERVED0 ;
 __O SPI_TBW_Typedef TBW;
 __I SPI_RBR_Typedef RBR;
 __IO SPI_IE_Typedef IE;
 __IO SPI_IF_Typedef IF;
 __I SPI_TB_Typedef TB;
 __I SPI_RB_Typedef RB;
 __I SPI_STA_Typedef STA;
} SPI_TypeDef;
#define APB_BASE (0x40000000UL)
#define SPI_BASE (APB_BASE + 0x08000)
#define SPI ((SPI_TypeDef *) SPI_BASE )
```

12. 3 宏定义

SPI 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_spi.h 中。

```
/* SPI使能控制 */
#define SPI_Enable()   (SPI->CON.EN = 1)
#define SPI_Disable() (SPI->CON.EN = 0)
/* SPI接收使能控制 */
```

```
#define SPI_RecEnable()   (SPI->CON.REN = 1)
#define SPI_RecDisable() (SPI->CON.REN = 0)
/* SPI软件复位 */
#define SPI_Rst() (SPI->CON.RST = 1)
```

12. 4 库函数

SPI 库函数定义于 lib_spi.c 中，声明于 lib_spi.h 中。

12. 4. 1 函数SPI_Init

- 函数原型：void SPI_Init(SPI_InitStruType* SPI_InitStruct)
- 功能描述：SPI 初始化
- 输入参数：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
typedef struct
{
    uint32_t   SPI_Freq;                //SPI频率
    SPI_TYPE_DFS   SPI_Df;              //通讯数据格式
    SPI_TYPE_MODE   SPI_Mode;           //通讯模式
    TYPE_FUNCEN   SPI_DelayRec;         //延时接收使能
    TYPE_FUNCEN   SPI_DelaySend;        //发送间隔使能
    uint8_t   SPI_SendDelayPeroid;      //发送间隔周期
}SPI_InitStruType;
```

通信数据格式枚举类型 SPI_TYPE_DFS：

枚举元素	数值	描述
SPI_RiseSendFallRec	0x0	上升沿发送（先），下降沿接收（后）
SPI_FallSendRiseRec	0x1	下降沿发送（先），上升沿接收（后）
SPI_RiseRecFallSend	0x2	上升沿接收（先），下降沿发送（后）
SPI_FallRecRiseSend	0x3	下降沿接收（先），上升沿发送（后）

表 12-1 SPI_TYPE_DFS

通讯模式选择枚举类型 SPI_TYPE_MODE：

枚举元素	数值	描述
SPI_Mode_Master	0x0	通讯模式：主控
SPI_Mode_Slave	0x1	通讯模式：从动

表 12-2 SPI_TYPE_MODE

12. 4. 2 函数SPI_ITConfig

- 函数原型：void SPI_ITConfig(SPI_TYPE_IT SPI_IE, TYPE_FUNCEN NewState)
- 功能描述：SPI 中断配置
- 输入参数：
  - SPI_IE：中断类型，详见表 12-3
  - NewState：使能、失能
- 返回值：无

中断选择枚举类型 SPI_TYPE_IT：

枚举元素	数值	描述
SPI_IT_TB	0x01	发送缓冲器空中断
SPI_IT_RB	0x02	接收缓冲器满中断
SPI_IT_TE	0x04	发送数据错误中断
SPI_IT_RO	0x08	接收数据溢出中断
SPI_IT_ID	0x10	空闲状态中断
SPI_IT_NSS	0x20	片选变化中断
SPI_IT_TBWE	0x40	发送数据写错误中断

表 12-3 SPI_TYPE_IT

12. 4. 3 函数SPI_DataFormatConfig

- 函数原型：void SPI_DataFormatConfig(SPI_TYPE_DFS Type)
- 功能描述：SPI 数据格式配置
- 输入参数：数据格式，详见表 12-1
- 返回值：无

12. 4. 4 函数SPI_Send

- 函数原型：
  - Void SPI_SendByte(uint8_t Temp)
  - Void SPI_SendHalfWord(uint16_t Temp)
  - Void SPI_SendWord(uint32_t Temp)
- 功能描述：SPI 发送字节、半字、字数据
- 输入参数：字节、半字、字数据
- 返回值：无

12. 4. 5 函数SPI_Rec

- 函数原型：
  - uint8_t SPI_RecByte(void)
  - uint16_t SPI_RecHalfWord(void)
  - uint32_t SPI_RecWord(void)
- 功能描述：SPI 接收字节、半字、字数据
- 输入参数：无
- 返回值：接收到的字节、半字、字数据

12. 4. 6 函数SPI_TBIMConfig

- 函数原型：void SPI_TBIMConfig(SPI_TYPE_TRBIM Type)
- 功能描述：SPI 发送缓冲器空中断模式选择
- 输入参数：空中断模式，详见
- 返回值：无

中断模式选择枚举类型 SPI_TYPE_TRBIM：

枚举元素	数值	描述
SPI_IType_BYTE	0x0	字节空中断
SPI_IType_HALFWORD	0x1	半字空中断
SPI_IType_WORD	0x2	字空中段

表 12-4 SPI_TYPE_TRBIM

12. 4. 7 函数SPI_RBIMConfig

- 函数原型：void SPI_RBIMConfig(SPI_TYPE_TRBIM Type)
- 功能描述：SPI 接收缓冲器满中断模式选择
- 输入参数：满中断模式，详见表 12-4
- 返回值：无

12. 4. 8 函数SPI_GetFlagStatus

- 函数原型：FlagStatus SPI_GetFlagStatus(SPI_TYPE_FLAG Flag)
- 功能描述：SPI 读取标志位状态
- 输入参数：标志位选择，详见表 12-5
- 返回值：SET/RESET

标志位选择枚举类型 SPI_TYPE_FLAG：

枚举元素	数值	描述
SPI_Flag_TB	0x00000001	发送缓冲器空中断标志
SPI_Flag_RB	0x00000002	接收缓冲器满中断标志
SPI_Flag_TE	0x00000004	发送错误中断标志（仅从动模式支持）
SPI_Flag_RO	0x00000008	接收数据溢出中断标志
SPI_Flag_ID	0x00000010	空闲中断标志（仅主控模式支持）
SPI_Flag_NSSIF	0x00000020	片选变化中断标志（仅从动模式支持）
SPI_Flag_TBWE	0x00000040	发送数据写错误中断标志
SPI_Flag_NSS	0x00000080	片选标志（仅从动模式支持）
SPI_Flag_TBEF0	0x00000100	TB0 空标志
SPI_Flag_TBEF1	0x00000200	TB1 空标志
SPI_Flag_TBEF2	0x00000400	TB2 空标志
SPI_Flag_TBEF3	0x00000800	TB3 空标志
SPI_Flag_RBFF0	0x00001000	RB0 满标志
SPI_Flag_RBFF1	0x00002000	RB1 满标志
SPI_Flag_RBFF2	0x00004000	RB2 满标志
SPI_Flag_RBFF3	0x00008000	RB3 满标志
SPI_Flag_IDLE	0x00010000	空闲标志（仅主控模式支持）
SPI_Flag_TMS	0x00020000	帧发送间隔状态标志（仅主控模式支持）

表 12-5　SPI_TYPE_FLAG

12. 4. 9　函数SPI_GetITStatus

- 函数原型：ITStatus　SPI_GetITStatus(SPI_TYPE_IT　Flag)
- 功能描述：检查中断状态,未使能相应中断时不会返回 SET
- 输入参数：中断标志位，详见表 12-3
- 返回值：SET/RESET

12. 4. 10 函数SPI_ClearITPendingBit

- 函数原型：void　SPI_ClearITPendingBit(SPI_CLR_IF　Flag)
- 功能描述：SPI 中断标志清除
- 输入参数：标志位清除选择，详见表 12-6
- 返回值：无

标志位清除选择枚举类型 SPI_CLR_IF：

枚举元素	数值	描述
SPI_Clr_TE	0x04	发送错误中断标志
SPI_Clr_RO	0x08	接收数据溢出中断标志
SPI_Clr_ID	0x10	空闲中断标志
SPI_Clr_NSS	0x20	片选标志
SPI_Clr_TBWE	0x40	发送数据写错误中断标志

表 12-6　SPI_CLR_IF

12. 5 函数库应用示例

```
/* SPI初始化 */
void User_SPIInit(void)
{
    SPI_InitStruType SPI_InitStruct;                    //定义结构体
    SPI_InitStruct.SPI_Mode = SPI_Mode_Master;          //模式设置
    SPI_InitStruct.SPI_Df = SPI_FallSendRiseRec;        //设置数据格式
    SPI_InitStruct.SPI_Freq = 80000;                    //波特率设置
    SPI_InitStruct.SPI_DelayRec = Enable;               //接收延时
    SPI_InitStruct.SPI_DelaySend = Disable;             //发送延时
    SPI_Init(&SPI_InitStruct);                          //初始化SPI
    SPI_Enable();                                       //使能SPI
}
```

第13章 FLASH存储器自编程（IAP）

13. 1 功能概述

- 进行 IAP 操作前需先进行解锁，去除相关寄存器的写保护
- 支持页擦除模式
- 支持字编程模式，每个字包括 4 个字节
- IAP 自编程操作程序需放在芯片的 SRAM 中执行，并在程序中对 FLASH 擦除或编程结果进行校验
- IAP 操作过程中可以响应中断

13. 2 寄存器结构

IAP 的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
 __IO IAP_CON_Typedef CON;
 __IO IAP_ADDR_Typedef ADDR;
 __IO IAP_DATA_Typedef DATA;
 __IO IAP_TRIG_Typedef TRIG;
 __IO IAP_UL_Typedef UL;
 __IO IAP_STA_Typedef STA;
} IAP_TypeDef;

#define APB_BASE (0x40000000UL)
#define IAP_BASE (APB_BASE + 0x00800)
#define IAP ((IAP_TypeDef *) IAP_BASE )
```

13. 3 宏定义

IAP 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_flashiap.h 中。

```
/* 寄存器解锁 */
#define FlashIAP_RegUnLock() (IAP->UL.IAPUL = 0x000000A5)
#define FlashIAP_RegLock()    (IAP->UL.IAPUL = 0x0)

/* 使能IAP */
#define FlashIAP_Enable()   (IAP->CON.EN = 0x1)
#define FlashIAP_Disable() (IAP->CON.EN = 0x0)

/* 访问IAP请求 */
#define FlashIAP_REQ() (IAP->CON.FLASH_REQ = 0x1)
```

13. 4 库函数

IAP 库函数定义于 lib_flashiap.c 中，声明于 lib_flashiap.h 中。

13. 4. 1 函数Flashlap_Unlock

- 函数原型：ErrorStatus Flashlap_Unlock(void)
- 功能描述：IAP 解锁与访问请求
- 输入参数：无
- 返回值：成功、失败

13. 4. 2 函数Flashlap_WriteEnd

- 函数原型：ErrorStatus Flashlap_WriteEnd(void)
- 功能描述：IAP 写结束
- 输入参数：无
- 返回值：成功、失败

13. 4. 3 函数Flashlap_ErasePage

- 函数原型：ErrorStatus Flashlap_ErasePage(uint8_t Page_Addr)
- 功能描述：IAP 页擦除
- 输入参数：8 位页地址
- 返回值：成功、失败

13. 4. 4 函数Flashlap_WriteCont

- 函数原型：ErrorStatus Flashlap_WriteCont(uint8_t Unit_addr, uint8_t Page_addr, uint32_t Data32)
- 功能描述：Flash 连续写（内部调用）
- 输入参数：
  - Unit_addr：单元地址
  - Page_addr：页地址
  - Data32：数据
- 返回值：成功、失败

13. 4. 5 函数FlashIap_WriteWord

- 函数原型：ErrorStatus FlashIap_WriteWord(uint8_t Unit_addr, uint8_t Page_addr, uint32_t Data32)
- 功能描述：Flash 写一个字
- 输入参数：
  - Unit_addr：单元地址
  - Page_addr：页地址
  - Data32：数据
- 返回值：成功、失败

13. 4. 6 函数Flash_Read

- 函数原型：ErrorStatus Flash_Read(uint32_t* Ram_Addr, uint32_t Flash_Addr, uint8_t Len)
- 功能描述：Flash 读数据
- 输入参数：
  - Ram_Addr：读出数据的存放地址
  - Flash_Addr：要读取 Flash 的起始地址
  - Len：读取的字长度
- 返回值：成功、失败

13. 5 函数库应用示例

```
/* Flash IAP操作示例 */
uint32_t Flash_WriteBuf;                    //定义写数据缓存
uint32_t Flash_ReadBuf;                     //定义读数据缓存

/* 判断是否写成功 */
if(FlashIap_WriteWord(0,90, Flash_ WriteBuf) == ERROR)
{                                           //向第90页第0单元写一个字的数据
    return ERROR;
}
if(Flash_Read((uint8_t *)(& Flash_ReadBuf),92160,4) == ERROR)
{                                           //读出写的字数据
    return ERROR;
}
```

第14章　看门狗定时器（WDT）

14. 1 功能概述

- 支持寄存器写保护
- 可选时钟源
- 可配置复位使能与中断使能

14. 2 特殊说明

WDT 模块的所有寄存器都受到了写保护。因此，所有对 WDT 模块的操作都需要先调用“WDT_RegUnLock()”宏来解除写保护，操作完成后调用“WDT_RegLock()”宏来使能写保护。

14. 3 寄存器结构

WDT 的寄存器定义于文件 ES8H296.h。

```
typedef struct
{
 __O WDT_LOAD_Typedef LOAD;
 __I WDT_VALUE_Typedef VALUE;
 __IO WDT_CON_Typedef CON;
 __O WDT_INTCLR_Typedef INTCLR;
 __I WDT_RIS_Typedef RIS;
 uint32_t RESERVED0[59] ;
 __IO WDT_LOCK_Typedef LOCK;
} WDT_TypeDef;

#define APB_BASE (0x40000000UL)
#define WDT_BASE (APB_BASE + 0x01C00)
#define WDT ((WDT_TypeDef *) WDT_BASE )
```

14. 4 宏定义

WDT 的一些功能使用宏定义的方法来定义，这些宏定义在文件 lib_wdt.h 中。

```
/*  寄存器写保护控制  */
#define WDT_RegUnLock() (WDT->LOCK.Word = 0x1ACCE551)
#define WDT_RegLock()     (WDT->LOCK.Word = 0x0)

/* WDT使能控制  */
#define WDT_Enable()    (WDT->CON.EN = 1)
#define WDT_Disable() (WDT->CON.EN = 0)
```

```
/* WDT清狗 */
#define WDT_Clear(){WDT_RegUnLock();WDT->INTCLR.INTCLR=0;WDT_RegLock();}

/* WDT中断使能控制 */
#define WDT_ITEnable()   (WDT->CON.IE = 1)
#define WDT_ITDisable() (WDT->CON.IE = 0)

/* WDT复位使能控制 */
#define WDT_RstEnable()   (WDT->CON.RSTEN = 1)
#define WDT_RstDisable() (WDT->CON.RSTEN = 0)

/* WDT计数时钟选择 */
#define WDT_CLOCK_PCLK()    (WDT->CON.CLKS = 0)
#define WDT_CLOCK_WDT()          (WDT->CON.CLKS = 1)
```

14. 5 库函数

WDT 库函数定义于 lib_wdt.c 中，声明于 lib_wdt.h 中。

14. 5. 1 函数WDT_Init

- 函数原型：void WDT_Init(WDT_InitStruType* WDT_InitStruct)
- 功能描述：WDT 初始化
- 输入参数：初始化配置结构体地址
- 返回值：无

初始化配置结构原型：

```
/* WDT初始化配置结构体定义 */
typedef struct
{
    uint32_t   WDT_Tms;                    //定时时间，单位ms
    TYPE_FUNCEN   WDT_IE;                  //中断使能
    TYPE_FUNCEN   WDT_Rst;                 //复位使能
    WDT_TYPE_CLKS   WDT_ClockS;            //时钟选择
}WDT_InitStruType;
```

时钟选择枚举类型 WDT_TYPE_CLKS：

枚举元素	数值	描述
WDT_ClockS_Pclk	0x0	时钟选择:PCLK
WDT_ClockS_WDT	0x1	时钟选择:WDT(约 32Khz)

表 14-1 WDT_TYPE_CLKS

14. 5. 2 函数WDT_SetReloadValue

- 函数原型：void WDT_SetReloadValue(uint32_t Value)
- 功能描述：设置 WDT 计数重装初值
- 输入参数：32 位无符号整型数值
- 返回值：无

14. 5. 3 函数WDT_GetValue

- 函数原型：uint32_t WDT_GetValue(void)
- 功能描述：获取 WDT 目前计数值
- 输入参数：无
- 返回值：32 位无符号整型数值

14. 5. 4 函数WDT_GetFlagStatus

- 函数原型：FlagStatus WDT_GetFlagStatus(void)
- 功能描述：获取 WDT 中断标志位
- 输入参数：无
- 返回值：SET/RESET

14. 6 函数库应用示例

```
void User_WDTInit(void)
{
    WDT_InitStruType x;                          //定义结构
    WDT_RegUnLock();                             //解锁写保护
    x.WDT_Tms = 10;                              //定时10ms
    x.WDT_IE = DISABLE;                          //中断设置
    x.WDT_Rst = DISABLE;                         //复位设置
    x.WDT_ClockS = WDT_ClockS_WDT;               //时钟源选择
    WDT_Init(&x);                                //初始化WDT
    WDT_Enable();                                //使能WDT
}
```

第15章 波特率误差

在 UART、IIC、SPI 模块的库函数中，用户可直接指定通讯波特率。但是由于硬件的实现方式，所得到的真实波特率与用户所指定的波特率可能会存在误差。

15. 1 UART波特率误差

误差可按照以下步骤计算：

1. 计算 BRR 寄存器值

$$BRR = INT（\frac{Fpclk}{Dbaud \times n} - 1）$$

若 BRR>2047，则 BRR=2047。

其中，Fpclk 为系统频率（Hz），Dbaud 为用户设置的目标波特率（Hz），n 的取值与 UART_ClockSet 变量有关，若 UART_ClockSet=UART_Clock_1，n=16；若 UART_ClockSet=UART_Clock_2，n=32；若 UART_ClockSet=UART_Clock_3，n=64。

2. 计算真实波特率

$$Rbaud = \frac{Fpclk}{(BRR + 1) \times n}$$

其中，Fpclk 与 n 的取值与上述相同。BRR 为上述公式中计算所得的值。

3. 计算误差

$$误差 = \frac{Rbaud - Dbaud}{Dbaud} \times 100\%$$

下表为 Fpclk 为 16MHz 时，一些典型波特率的误差。

	UART_Clock_1			UART_Clock_2			UART_Clock_3		
Dbaud	BRR	Rbaud	误差	BRR	Rbaud	误差	BRR	Rbaud	误差
115200	7	125000	9%	3	125000	9%	1	125000	9%
57600	16	58823	2%	7	62500	9%	3	62500	9%
38400	25	38461	0%	12	38461	0%	5	41666	9%
19200	51	19230	0%	25	19230	0%	12	19230	0%
14400	68	14492	1%	33	14705	2%	16	14705	2%
9600	103	9615	0%	51	9615	0%	25	9615	0%
7200	137	7246	1%	68	7246	1%	33	7352	2%
4800	207	4807	0%	103	4807	0%	51	4807	0%
3600	276	3610	0%	137	3623	1%	68	3623	1%
2400	415	2403	0%	207	2403	0%	103	2403	0%
1800	554	1801	0%	276	1805	0%	137	1811	1%
1200	832	1200	0%	415	1201	0%	207	1201	0%
600	1665	600	0%	832	600	0%	415	600	0%
300	2047	488	63%	1665	300	0%	832	300	0%

150	2047	488	225%	2047	244	63%	1665	150	0%

表 15-1　UART 波特率误差

15. 2 IIC波特率误差

误差可按照以下步骤计算：

1. 计算 TJP 寄存器值

$$TJP = INT（\frac{Fpclk}{Dbaud \times n} - 1）$$

若 TJP>255，则 TJP=255。

其中，Fpclk 为系统频率（Hz），Dbaud 为用户设置的目标波特率（Hz），n 的取值与 IIC_16Xsamp 参数有关，若 IIC_16XSamp=Disable，n=16；若 IIC_16XSamp=Enable，n=24。

2. 计算真实波特率

$$Rbaud = \frac{Fpclk}{(TJP + 1) \times n}$$

其中，Fpclk 与 n 的取值与上述相同。TJP 为上述公式中计算所得的值。

3. 计算误差

$$误差 = \frac{Rbaud - Dbaud}{Dbaud} \times 100\%$$

下表为 Fpclk 为 16MHz 时，一些典型波特率的误差。

IIC_16XSamp=Disable				IIC_16XSamp=Enable			
Dbaud	**TJP**	**Rbaud**	**误差**	**Dbaud**	**TJP**	**Rbaud**	**误差**
400000	1	500000	25%	400000	0	666666	67%
350000	1	500000	43%	350000	0	666666	90%
300000	2	333333	11%	300000	1	333333	11%
250000	3	250000	0%	250000	1	333333	33%
200000	4	200000	0%	200000	2	222222	11%
150000	5	166666	11%	150000	3	166666	11%
100000	9	100000	0%	100000	5	111111	11%
80000	11	83333	4%	80000	7	83333	4%
60000	15	62500	4%	60000	10	60606	1%
50000	19	50000	0%	50000	12	51282	3%
40000	24	40000	0%	40000	15	41666	4%
20000	49	20000	0%	20000	32	20202	1%
10000	99	10000	0%	10000	65	10101	1%
5000	199	5000	0%	5000	132	5012	0%
1000	255	3906	291%	1000	255	2604	160%

表 15-2　IIC 波特率误差

15. 3 SPI波特率误差

误差可按照以下步骤计算：

1. 计算 CKS 寄存器值

$$CKS = INT（\frac{Fpclk}{Dbaud \times 2}）$$

若 CKS>255，则 CKS=255。

其中，Fpclk 为系统频率（Hz），Dbaud 为用户设置的目标波特率（Hz）。

2. 计算真实波特率

$$Rbaud = \frac{Fpclk}{CKS \times 2}$$

其中，Fpclk 的取值与上述相同。CKS 为上述公式中计算所得的值。

3. 计算误差

$$误差 = \frac{Rbaud - Dbaud}{Dbaud} \times 100\%$$

下表为 Fpclk 为 16MHz 时，一些典型波特率的误差。

Fpclk=16 000 000			
Dbaud	**CKS**	**Rbaud**	**误差**
16000000	0	16000000	0%
15000000	0	16000000	7%
10000000	0	16000000	60%
8000000	1	8000000	0%
6000000	1	8000000	33%
4000000	2	4000000	0%
2000000	4	2000000	0%
1000000	8	1000000	0%
800000	10	800000	0%
600000	13	615384	3%
400000	20	400000	0%
200000	40	200000	0%
100000	80	100000	0%
80000	100	80000	0%
60000	133	60150	0%
40000	200	40000	0%
20000	255	31372	57%

表 15-3　SPI 波特率误差