

文档编号: AN_074

上海东软载波微电子有限公司

用户手册

HR8P506 库函数

修订历史

版本	修订日期	修改概要
V1.0	2016-02-26	初版发布
V1.1	2016-12-02	1. 更新了定时器的库函数 2. 增加了 LED 库函数章节
V1.2	2017-4-5	Library 中增加了 IAPOP 读的例程，AN073 IAP 改为 IAPOP 方式。
V1.3	2018-1-17	1. 将 EUART 库函数中以 U7816 开头的部分函数名改为以 EUART 开头。 2. 修改 printf 函数相关描述。

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

版权所有©

上海东软载波微电子有限公司

本资料内容为上海东软载波微电子有限公司在现有数据资料基础上慎重且力求准确无误编制而成，本资料中所记载的实例以正确的使用方法和标准操作为前提，使用方在应用该等实例时请充分考虑外部诸条件，上海东软载波微电子有限公司不担保或确认该等实例在使用方的适用性、适当性或完整性，上海东软载波微电子有限公司亦不对使用方因使用本资料所有内容而可能或已经带来的风险或后果承担任何法律责任。基于使本资料的内容更加完善等原因，上海东软载波微电子有限公司保留未经预告的修改权。使用方如需获得最新的产品信息，请随时用上述联系方式与上海东软载波微电子有限公司联系。

目 录

第 1 章	概述	9
1.1	关于本文档	9
1.2	芯片简介	9
1.3	芯片系统时钟	12
1.4	文档规范	12
1.4.1	缩写	13
1.4.2	命名规则	13
1.4.3	数据类型	13
第 2 章	开始使用	15
2.1	文件结构	15
2.2	函数库的配置	15
2.2.1	选择 printf 函数使用的串口	15
2.2.2	函数库的引用	16
2.2.3	中断函数	16
第 3 章	系统控制单元 SCU	18
3.1	功能概述	18
3.2	特殊说明	18
3.3	寄存器结构	18
3.4	宏定义	19
3.5	库函数	21
3.5.1	SCU_NMISelect 函数	21
3.5.2	SCU_GetPWRCFlagStatus 函数	22
3.5.3	SCU_ClearPWRCFlagBit 函数	22
3.5.4	SCU_GetLVDFlagStatus 函数	22
3.5.5	SCU_SysClkSelect 函数	22
3.5.6	SCU_GetSysClk 函数	23
3.5.7	DeviceClockAllEnable 函数	23
3.5.8	DeviceClockAllDisable 函数	23
3.5.9	SCU_OpenXTAL 函数	23
3.5.10	DeviceClock_Config 函数	23
3.5.11	PLLClock_Config 函数	24
3.6	库函数应用示例	24
第 4 章	内核模块	25
4.1	功能概述	25
4.2	寄存器结构	25
4.3	宏定义	25
4.4	库函数	25
4.4.1	NVIC_EnableIRQ 函数	25
4.4.2	NVIC_DisableIRQ 函数	26
4.4.3	NVIC_SetPriority 函数	27
4.4.4	NVIC_GetPriority 函数	27
4.5	库函数应用示例	27

第 5 章	通用输入输出 GPIO.....	28
5.1	功能概述	28
5.2	寄存器结构	28
5.3	宏定义.....	29
5.4	库函数.....	29
5.4.1	GPIO 模块的 API	29
5.4.2	外部端口模块的 API.....	33
5.4.3	外部按键模块的 API.....	34
5.5	库函数应用示例	35
第 6 章	定时器/计数器 T16N/T32N.....	37
6.1	功能概述	37
6.1.1	T16N	37
6.1.2	T32N	37
6.2	寄存器结构	38
6.3	宏定义.....	39
6.4	库函数.....	39
6.4.1	16 位定时器的 API	39
6.4.2	32 位定时器的 API	45
6.5	库函数应用示例	49
6.5.1	定时器	49
6.5.2	计数器	51
第 7 章	数模转换 ADC	53
7.1	功能概述	53
7.2	寄存器结构	53
7.3	宏定义.....	53
7.4	库函数.....	54
7.4.1	ADC_Init 函数	54
7.4.2	ADC_ACPConfig 函数	56
7.4.3	ADC_Start 函数.....	57
7.4.4	ADC_SoftStart 函数	57
7.4.5	ADC_SoftStop 函数.....	57
7.4.6	ADC_GetConvValue 函数.....	57
7.4.7	ADC_GetConvStatus 函数.....	57
7.4.8	ADC_GetACPMeanValue 函数.....	57
7.4.9	ADC_GetIFStatus 函数	58
7.4.10	ADC_ClearIFStatus 函数	58
7.4.11	ADC_Reset 函数.....	58
7.5	库函数应用示例	58
第 8 章	液晶显示控制器 LCDC.....	60
8.1	功能概述	60
8.2	寄存器结构	60
8.3	宏定义.....	60
8.4	库函数.....	60
8.4.1	LCD_Init 函数.....	61

8.4.2	LCD_GayscaleConfig 函数	63
8.4.3	LCD_FlickerTimeConfig 函数	64
8.4.4	LCD_PixelWriteByte 函数	64
8.4.5	LCD_PixelWriteHalfWord 函数	65
8.4.6	LCD_PixelWriteWord 函数	65
8.4.7	LCD_Reset 函数	65
8.5	库函数应用示例	66
第 9 章	数码管显示控制器(LEDIC)	67
9.1	功能概述	67
9.2	寄存器结构	67
9.3	宏定义	67
9.4	库函数	67
9.4.1	LED_Init 函数	67
9.4.2	LED_PixelWriteWord 函数	68
9.4.3	LED_Reset 函数	68
9.5	库函数应用示例	68
第 10 章	通用异步收发器 UART	70
10.1	功能概述	70
10.2	寄存器结构	70
10.3	宏定义	71
10.4	库函数	71
10.4.1	UART_Init 函数	71
10.4.2	UART_ITConfig 函数	72
10.4.3	UART_TBIMConfig 函数	73
10.4.4	UART_RBIMConfig 函数	73
10.4.5	UART_Send 函数	73
10.4.6	UART_Rcv 函数	74
10.4.7	UART_GetStatus 函数	74
10.4.8	UART_GetFlagStatus 函数	74
10.4.9	UART_GetITStatus 函数	75
10.4.10	UART_ClearITPendingBit 函数	75
10.5	库函数应用示例	76
第 11 章	增强型通用异步收发器 EUART	78
11.1	功能概述	78
11.2	寄存器结构	78
11.3	宏定义	78
11.4	库函数	79
11.4.1	EUART_ModeConfig 函数	79
11.4.2	EUART_Init 函数	79
11.4.3	EUART_BaudConfig 函数	80
11.4.4	EUART_ITConfig 函数	80
11.4.5	EUART_TBIMConfig 函数	81
11.4.6	EUART_RBIMConfig 函数	81
11.4.7	EUART_GetFlagStatus 函数	81

11. 4. 8	EUART_GetITStatus 函数	82
11. 4. 9	EUART_ClearITPendingBit 函数	82
11. 4. 10	U7816_Init 函数	82
11. 4. 11	U7816_EIOChConfig 函数	84
11. 4. 12	U7816_EIODirection 函数	84
11. 4. 13	U7816_Send 函数	84
11. 4. 14	U7816_Rcv 函数	84
11. 5	库函数应用示例	85
第 12 章	IIC 串行总线	86
12. 1	功能概述	86
12. 2	寄存器结构	86
12. 3	宏定义	87
12. 4	库函数	88
12. 4. 1	IIC_Init 函数	88
12. 4. 2	IIC_ITConfig 函数	89
12. 4. 3	IIC_SendAddress 函数	89
12. 4. 4	IIC_SetAddress 函数	90
12. 4. 5	IIC_RecModeConfig 函数	90
12. 4. 6	IIC_TBIMConfig 函数	90
12. 4. 7	IIC_RBIMConfig 函数	90
12. 4. 8	IIC_AckDelay 函数	91
12. 4. 9	IIC_TISConfig 函数	91
12. 4. 10	IIC_Send 函数	92
12. 4. 11	IIC_Rcv 函数	92
12. 4. 12	IIC_GetRWMode 函数	92
12. 4. 13	IIC_GetTBStatus 函数	92
12. 4. 14	IIC_GetFlagStatus 函数	93
12. 4. 15	IIC_GetITStatus 函数	93
12. 4. 16	IIC_ClearITPendingBit 函数	94
12. 5	库函数应用示例	94
第 13 章	SPI 串行总线	95
13. 1	功能概述	95
13. 2	寄存器结构	95
13. 3	宏定义	95
13. 4	库函数	96
13. 4. 1	SPI_Init 函数	96
13. 4. 2	SPI_ITConfig 函数	97
13. 4. 3	SPI_DataFormatConfig 函数	97
13. 4. 4	SPI_Send 函数	97
13. 4. 5	SPI_Rcv 函数	97
13. 4. 6	SPI_TBIMConfig 函数	98
13. 4. 7	SPI_RBIMConfig 函数	98
13. 4. 8	SPI_GetFlagStatus 函数	98
13. 4. 9	SPI_GetITStatus 函数	99

13.4.10	SPI_GetStatus 函数	99
13.4.11	SPI_ClearITPendingBit 函数	100
13.4.12	Clear_TBW 函数	100
13.4.13	Clear_RBR 函数	100
13.5	库函数应用示例	100
第 14 章	FLASH 存储器自编程 IAP	102
14.1	寄存器结构	102
14.2	寄存器结构	102
14.3	宏定义	102
14.4	库函数	103
14.4.1	IAP_Unlock 函数	103
14.4.2	IAP_WriteEnd 函数	103
14.4.3	IAP_ErasePage 函数	103
14.4.4	IAP_WriteCont 函数	103
14.4.5	IAP_WriteWord 函数	103
14.4.6	IAP_Read 函数	104
14.5	库函数应用示例	104
第 15 章	看门狗定时器 WDT	105
15.1	功能概述	105
15.2	特殊说明	105
15.3	寄存器结构	105
15.4	宏定义	105
15.5	库函数	106
15.5.1	WDT_Init 函数	106
15.5.2	WDT_SetReloadValue 函数	106
15.5.3	WDT_GetValue 函数	106
15.5.4	WDT_GetFlagStatus 函数	106
15.6	库函数应用示例	107
第 16 章	实时时钟 RTC	108
16.1	寄存器结构	108
16.2	寄存器结构	108
16.3	宏定义	108
16.4	库函数	108
16.4.1	RTC_Init 函数	109
16.4.2	RTC_ReadSecond 函数	109
16.4.3	RTC_ReadMinute 函数	109
16.4.4	RTC_ReadHour 函数	109
16.4.5	RTC_ReadDay 函数	109
16.4.6	RTC_ReadMonth 函数	109
16.4.7	RTC_ReadYear 函数	110
16.4.8	RTC_WriteSecond 函数	110
16.4.9	RTC_WriteMinute 函数	110
16.4.10	RTC_WriteHour 函数	110
16.4.11	RTC_WriteDay 函数	110

16. 4. 12	RTC_WriteMonth 函数	110
16. 4. 13	RTC_WriteYear 函数	111
16. 4. 14	RTC_InterruptEnable 函数	111
16. 4. 15	RTC_InterruptDisable 函数	111
16. 4. 16	RTC_GetITFlag 函数	111
16. 4. 17	RTC_ClearAllITFlag 函数	112
16. 5	库函数应用示例	112
第 17 章	波特率误差	114
17. 1	UART 波特率误差	114
17. 2	IIC 波特率误差	115
17. 3	SPI 波特率误差	116

第1章 概述

1.1 关于本文档

本文档是 HR8P506 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，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，12 位 ADC，以及用于系统电源监测的 LVD 模块等外设。

◆工作条件

- ◇ 工作电压范围：2.2V ~ 5.5V
- ◇ 工作温度范围：-40 ~ 85℃（工业级）
- ◇ 工作主时钟频率：32KHz~48MHz
- ◇ 工作电流：I_{vdd} = 3.5mA（@内部 HRC 16MHz，典型值）
- ◇ 待机电流：I_{vdd} = 5uA（常温，典型值）

◆封装

- ◇ LQFP48 封装（支持 46 个 I/O 端口）
- ◇ LQFP44 封装（支持 42 个 I/O 端口）
- ◇ LQFP32/QFN32 封装（支持 30 个 I/O 端口）
- ◇ SOP28 封装（支持 26 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

◆复位

- ◇ 内嵌上电复位电路 POR
- ◇ 内嵌掉电复位电路 BOR
- ◇ 支持外部复位

◆时钟

- ◇ 外部晶体振荡器可配置，支持低速振荡器 32KHz 和高速振荡器 1~20MHz，可配置为系统时钟源
- ◇ 内部 16MHz RC 振荡器（HRC）可配置为系统时钟源，出厂前已校准（全温度，全电压范围内 16MHz 频率精度为±3%）
- ◇ 内部 32KHz RC 振荡器（LRC）作为 WDT 时钟源，可配置为系统时钟源
- ◇ 支持 PLL 倍频，时钟源可选择，最大可倍频至 48MHz，可配置为系统时钟源

◆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）
- ◇ 支持两组 SWD 调试接口可选择，通过配置字 DEBUG_S 进行选择
- ◇ 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
 - ◇ 36K 字节 FLASH 存储器
 - 支持 ISP 在线串行编程
 - 支持两组 ISP 编程接口可选择, 硬件自动识别有效的 ISP 编程接口
 - 支持 IAP 在应用中编程, 可选取部分区域作为数据存储使用
 - 支持 FLASH 编程代码加密保护
 - ◇ 8K 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
 - ◇ 最多 46 个双向 I/O 端口
 - PA 端口 (PA0~PA31)
 - PB 端口 (PB0~PB13)
 - ◇ 支持 8 路外部中断输入, 触发方式可配置, 每个 I/O 端口均可作为外部中断输入源
 - ◇ 支持 1 路按键中断输入, 触发方式可配置, 每个 I/O 端口均可作为按键中断输入源
- ◆ 定时器/计数器
 - ◇ T16N0: 16 位定时器/计数器, 带预分频器, 扩展输入捕捉/输出调制功能
 - ◇ T16N1: 16 位定时器/计数器, 带预分频器, 扩展输入捕捉/输出调制功能
 - ◇ T16N2: 16 位定时器/计数器, 带预分频器, 扩展输入捕捉/输出调制功能
 - ◇ T16N3: 16 位定时器/计数器, 带预分频器, 扩展输入捕捉/输出调制功能
 - ◇ T32N0: 32 位定时器/计数器, 带预分频器, 扩展输入捕捉/输出调制功能
 - ◇ RTC : 一路 RTC 实时时钟
- ◆ UART 通信接口
 - ◇ 支持二路 UART 通信接口 UART0, UART1
 - ◇ 支持全/半双工异步通信模式
 - ◇ 支持传输波特率可配置
 - ◇ 支持 8 级发送/接收缓冲器
 - ◇ 支持 7/8/9 位数据格式可配
 - ◇ 支持奇偶校验功能可配, 支持硬件自动奇偶校验位判断
 - ◇ 支持空闲帧检测
 - ◇ 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 PWM 调制输出, 且 PWM 占空比线性可调
 - ◇ 支持接收端口红外唤醒功能
 - ◇ 支持 UART 输入输出通讯端口极性可配置
- ◆ EUART 通信接口
 - ◇ 支持一路 EUART 通信接口 EUART0
 - ◇ 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

- ◇ 支持一路通信接口 I2C0
- ◇ 支持主控和从动模式
- ◇ 支持标准 I2C 总线协议，最高传输速率 400K bit/s
- ◇ 支持 7 位寻址方式
- ◇ 约定数据从最高位开始接收/发送
- ◇ 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

- ◇ 支持二路通信接口 SPI0, SPI1
- ◇ 支持主控模式和从动模式
- ◇ 支持 4 种通信数据格式
- ◇ 支持 4 级接收/发送缓冲器
- ◇ 支持数据接收和发送中断

◆ADC 模数转换器

- ◇ 支持 12 位转换结果，有效精度为 11 位
- ◇ 支持 16 通道模拟输入端
- ◇ 支持参考电压源可选择
- ◇ 支持中断产生
- ◇ 支持转换结果自动比较
- ◇ 支持定时触发 ADC 转换

◆LCDC 液晶显示控制器

- ◇ 支持最大 8 COM x 28SEG
- ◇ 支持时钟源可配置：LRC 的 4 分频，LOSC 的 4 分频或 PCLK 的 4096 分频
- ◇ 支持灰度调节功能
- ◇ 支持显示闪烁功能，闪烁频率可调
- ◇ 支持两种不同的 LCD 驱动波形
- ◇ 支持不同的偏置电压可调

◆LEDC 数码管显示控制器

- ◇ 支持 1~8 个 8 段式共阴极数码管
- ◇ 支持时钟源可配置：LRC 的 4 分频，LOSC 的 4 分频或 PCLK 的 4096 分频

◆RTC 实时时钟

- ◇ 仅 POR 上电复位有效，支持程序写保护，有效避免系统干扰对时钟造成的影响
- ◇ 采用外部 32.768KHz 晶体振荡器作为精确计时时钟源
- ◇ 可进行高精度数字校正，提供高精度计时
- ◇ 时钟调校提供两种时间精度，调校范围为 $\pm 384\text{ppm}$ （或 $\pm 128\text{ppm}$ ），可实现最大时间精度为 $\pm 1.5\text{ppm}$ （或 $\pm 0.5\text{ppm}$ ）
- ◇ 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 5 个可编程定时中断
- ◇ 提供 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，有效期至 2099 年
- ◇ 12 小时和 24 小时模式设置可选
- ◇ 低功耗设计：工作电压为 VDD=5.0V 时模块工作电流典型值为 0.5μA

1.3 芯片系统时钟

HR8P506 具有丰富的时钟及配置系统，详见下图：

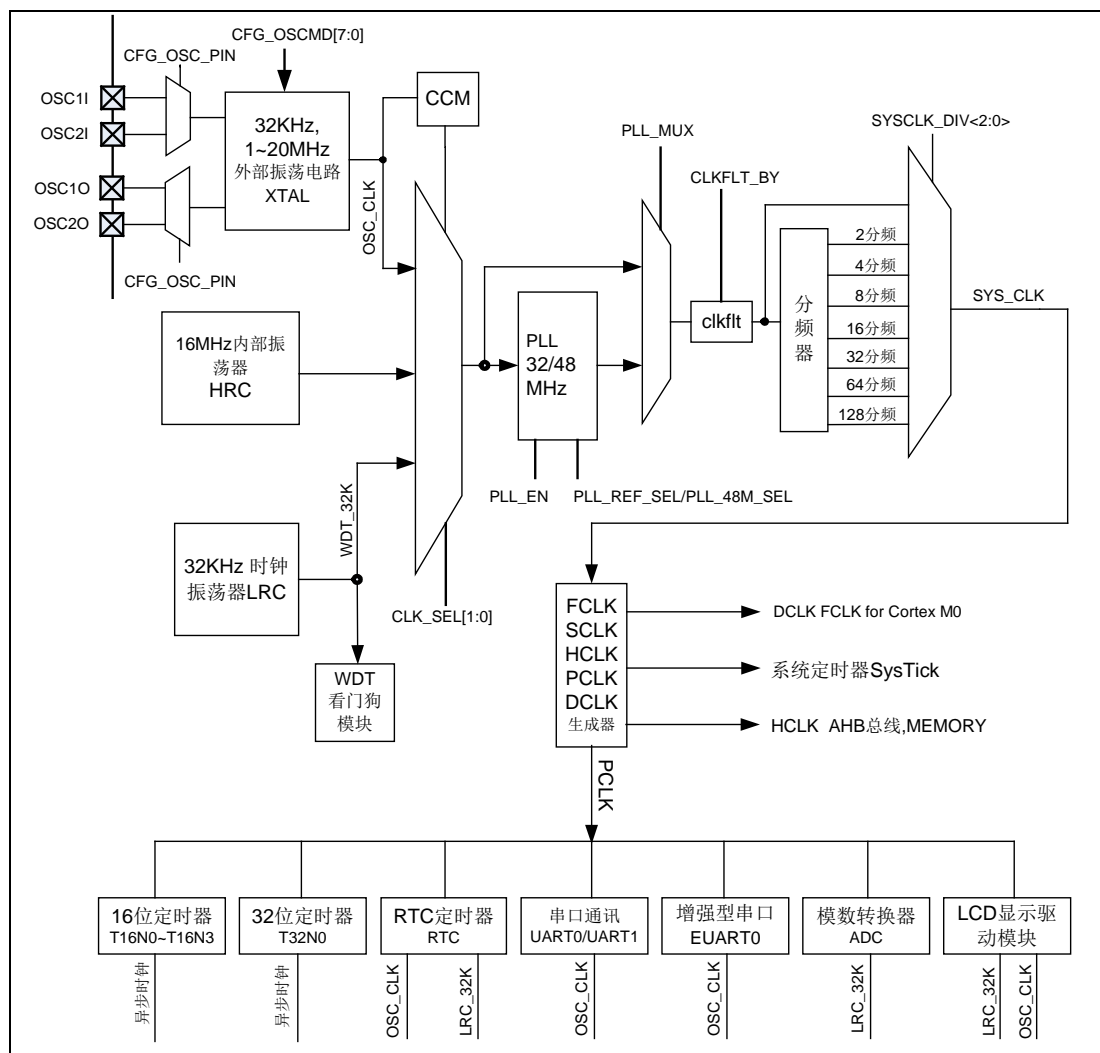


图 1-1 系统时钟

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 `type.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（SUCCESS）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。

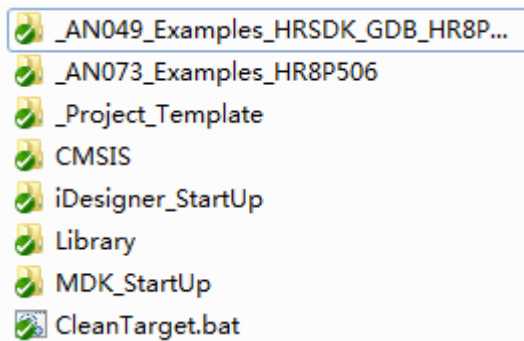


图 2-1 函数库文件夹结构

- ◆ **_AN073_Examples_HR8P506**
该文件夹下存放的是与底板相关的一些 demo。
- ◆ **_AN049_Examples_HR8P506**
该文件夹下存放的是库函数部分 damo。
- ◆ **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相关文件 system_HR8P506.c 和 system_HR8P506.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**
该文件夹下存放芯片的 iDesigner 启动文件 startup_HR8P506.S。
- ◆ **文件夹 MDK_StartUp**
该文件夹下存放芯片的 MDK 工程启动文件 startup_HR8P506.S。
- ◆ **ClearTarget.bat 文件**
该文件清除编译时产生的中间文件。

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_HR8P506.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_iic.h"
#include "lib_scs.h"
#include "lib_scu.h"
#include "lib_spi.h"
#include "lib_timer.h"
#include "lib_uart.h"
#include "lib_euart.h"
#include "lib_wdt.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。

2.2.3 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。外设相关的中断函数按照以下表格命名：

函数名	描述
Interrupt_PINT0_Handler	外部中断 0
Interrupt_PINT1_Handler	外部中断 1
Interrupt_PINT2_Handler	外部中断 2
Interrupt_PINT3_Handler	外部中断 3
Interrupt_PINT4_Handler	外部中断 4
Interrupt_PINT5_Handler	外部中断 5
Interrupt_PINT6_Handler	外部中断 6
Interrupt_PINT7_Handler	外部中断 7
Interrupt_T16N0_Handler	16 位定时器 0 中断
Interrupt_T16N1_Handler	16 位定时器 1 中断
Interrupt_T16N2_Handler	16 位定时器 2 中断
Interrupt_T16N3_Handler	16 位定时器 3 中断
Interrupt_T32N0_Handler	32 位定时器 0 中断
Interrupt_WDTINT_Handler	看门狗模块中断
Interrupt_RTCINT_Handler	RTC 模块中断
Interrupt_KINT_Handler	KINT 中断
Interrupt_ADCINT_Handler	ADC 模块中断
Interrupt_LVDINT_Handler	LCD 控制器模块中断

函数名	描述
Interrupt_PLLLKINT_Handler	PLLLK 中断
Interrupt_UART0_Handler	串口 0 中断
Interrupt_UART1_Handler	串口 1 中断
Interrupt_EUART0_Handler	增强型串口 0 中断
Interrupt_SPI0INT_Handler	SPI0 中断
Interrupt_SPI1INT_Handler	SPI1 中断
Interrupt_I2C0INT_Handler	IIC 模块中断
Interrupt_CCMINT_Handler	CCM 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
- ◇支持 PLL 时钟

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 HR8P506.h。

```
typedef struct
{
    __IO SCU_PROT_Typedef PROT;
    __IO SCU_NMICON_Typedef NMICON;
    __IO SCU_PWRC_Typedef PWRC;
    __IO SCU_FAULTFLAG_Typedef FAULTFLAG;
    uint32_t RESERVED0[4];
    __IO SCU_FLASHWAIT_Typedef FLASHWAIT;
    __IO SCU_SOFTCFG_Typedef SOFTCFG;
    __IO SCU_LVDCON_Typedef LVDCON;
    __IO SCU_CCM_Typedef CCM;
    __IO SCU_PLCLKCON_Typedef PLCLKCON;
    __IO SCU_TIMEREN_Typedef TIMEREN;
    __IO SCU_TIMERDIS_Typedef TIMERDIS;
    uint32_t RESERVED1;
    __IO SCU_SCLKEN0_Typedef SCLKEN0;
    __IO SCU_SCLKEN1_Typedef SCLKEN1;
    __IO SCU_PCLKEN_Typedef PCLKEN;
    __IO SCU_WAKEUPTIME_Typedef WAKEUPTIME;
    uint32_t RESERVED2[4];
    __IO SCU_TBLREMAPEN_Typedef TBLREMAPEN;
    __IO SCU_TBLOFF_Typedef TBLOFF;
} SCU_TypeDef;

#define APB_BASE (0x40000000UL)
#define SCU_BASE (APB_BASE + 0x000000)
#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->NMICON.NMIEN = 0x1)
#define SCU_NMI_Disable() (SCU->NMICON.NMIEN = 0x0)

/* -----LVD 模块----- */

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

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

/* LVD 触发电压选择 */
#define SCU_LVDVS_2V4()    (SCU->LVDCON.VS = 0x0)
#define SCU_LVDVS_2V6()    (SCU->LVDCON.VS = 0x1)
#define SCU_LVDVS_2V8()    (SCU->LVDCON.VS = 0x2)
#define SCU_LVDVS_3V()     (SCU->LVDCON.VS = 0x3)
#define SCU_LVDVS_3V2()    (SCU->LVDCON.VS = 0x4)
#define SCU_LVDVS_3V5()    (SCU->LVDCON.VS = 0x5)
#define SCU_LVDVS_3V7()    (SCU->LVDCON.VS = 0x6)
#define SCU_LVDVS_3V9()    (SCU->LVDCON.VS = 0x7)
#define SCU_LVDVS_4V2()    (SCU->LVDCON.VS = 0x8)
#define SCU_LVDVS_4V4()    (SCU->LVDCON.VS = 0x9)
#define SCU_LVDVS_4V6()    (SCU->LVDCON.VS = 0xA)
#define SCU_LVDVS_4V8()    (SCU->LVDCON.VS = 0xB)
#define SCU_LVDVS_5V()     (SCU->LVDCON.VS = 0xC)
#define SCU_LVDVS_5V2()    (SCU->LVDCON.VS = 0xD)
#define SCU_LVDVS_LVDIN() (SCU->LVDCON.VS = 0xE)

/* LVD 中断使能控制 */
#define SCU_LVDIT_Enable() (SCU->LVDCON.IE = 0x1)
#define SCU_LVDIT_Disable() (SCU->LVDCON.IE = 0x0)

/* LVD 中断标志位清除 */
#define SCU_LVDClearIFBit() (SCU->LVDCON.IF = 0)
```

/* LVD 中断产生模式选择 */

```
#define SCU_LVDIFS_Rise() (SCU->LVDCON.IFS = 0x0) //LVDO 上升沿产生中断
#define SCU_LVDIFS_Fall() (SCU->LVDCON.IFS = 0x1) //LVDO 下降沿产生中断
#define SCU_LVDIFS_High() (SCU->LVDCON.IFS = 0x2) //LVDO 高电平产生中断
#define SCU_LVDIFS_Low() (SCU->LVDCON.IFS = 0x3) //LVDO 低电平产生中断
#define SCU_LVDIFS_Change() (SCU->LVDCON.IFS = 0x4) //LVDO 电平变化产生中断
```

/* FLASH 访问等待时间选择 */

```
#define SCU_FlashWait_1Tclk() (SCU->FLASHWAIT.ACCT = 0x0)
#define SCU_FlashWait_2Tclk() (SCU->FLASHWAIT.ACCT = 0x1)
#define SCU_FlashWait_3Tclk() (SCU->FLASHWAIT.ACCT = 0x2)
#define SCU_FlashWait_4Tclk() (SCU->FLASHWAIT.ACCT = 0x3)
#define SCU_FlashWait_5Tclk() (SCU->FLASHWAIT.ACCT = 0x4)
#define SCU_FlashWait_6Tclk() (SCU->FLASHWAIT.ACCT = 0x5)
#define SCU_FlashWait_7Tclk() (SCU->FLASHWAIT.ACCT = 0x6)
#define SCU_FlashWait_8Tclk() (SCU->FLASHWAIT.ACCT = 0x7)
#define SCU_FlashWait_9Tclk() (SCU->FLASHWAIT.ACCT = 0x8)
#define SCU_FlashWait_10Tclk() (SCU->FLASHWAIT.ACCT = 0x9)
#define SCU_FlashWait_11Tclk() (SCU->FLASHWAIT.ACCT = 0xA)
#define SCU_FlashWait_12Tclk() (SCU->FLASHWAIT.ACCT = 0xB)
#define SCU_FlashWait_13Tclk() (SCU->FLASHWAIT.ACCT = 0xC)
#define SCU_FlashWait_14Tclk() (SCU->FLASHWAIT.ACCT = 0xD)
#define SCU_FlashWait_15Tclk() (SCU->FLASHWAIT.ACCT = 0xE)
#define SCU_FlashWait_16Tclk() (SCU->FLASHWAIT.ACCT = 0xF)
```

/* 系统时钟后分频选择 */

```
#define SCU_SysClk_Div1() (SCU->SCLKEN0.SYSCLKDIV = 0)
#define SCU_SysClk_Div2() (SCU->SCLKEN0.SYSCLKDIV = 1)
#define SCU_SysClk_Div4() (SCU->SCLKEN0.SYSCLKDIV = 2)
#define SCU_SysClk_Div8() (SCU->SCLKEN0.SYSCLKDIV = 3)
#define SCU_SysClk_Div16() (SCU->SCLKEN0.SYSCLKDIV = 4)
#define SCU_SysClk_Div32() (SCU->SCLKEN0.SYSCLKDIV = 5)
#define SCU_SysClk_Div64() (SCU->SCLKEN0.SYSCLKDIV = 6)
#define SCU_SysClk_Div128() (SCU->SCLKEN0.SYSCLKDIV = 7)
```

/* 外部时钟低功耗模式 */

```
#define SCU_XTAL_LP_Enable() (SCU->SCLKEN0.XTAL_LP = 0)
#define SCU_XTAL_LP_Disable() (SCU->SCLKEN0.XTAL_LP = 1)
```

/* 中断向量表重映射使能控制 */

```
#define SCU_TBLRemap_Enable() (SCU->TBLREMAPEN.EN= 1)
#define SCU_TBLRemap_Disable() (SCU->TBLREMAPEN.EN= 0)
```

/* 中断向量表偏移寄存器 x 最大为 $2^{24}=16777216$ */

#define SCU_TBL_Offset(x) (SCU->TBLOFF.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 SCU_NMISelect函数

◆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:

NMI_Type: 不可屏蔽中断类型

◆返回值: 无

typedef enum

```
{
    SCU_NMIIRQ_PINT0    = 0,      /* 外部端口中断 0 */
    SCU_NMIIRQ_PINT1    = 1,      /* 外部端口中断 1 */
    SCU_NMIIRQ_PINT2    = 2,      /* 外部端口中断 2 */
    SCU_NMIIRQ_PINT3    = 3,      /* 外部端口中断 3 */
    SCU_NMIIRQ_PINT4    = 4,      /* 外部端口中断 4 */
    SCU_NMIIRQ_PINT5    = 5,      /* 外部端口中断 5 */
    SCU_NMIIRQ_PINT6    = 6,      /* 外部端口中断 6 */
    SCU_NMIIRQ_PINT7    = 7,      /* 外部端口中断 7 */
    SCU_NMIIRQ_T16N0    = 8,      /* 16 位定时器/计数器 0 中断 */
    SCU_NMIIRQ_T16N1    = 9,      /* 16 位定时器/计数器 1 中断 */
    SCU_NMIIRQ_T16N2    = 10,     /* 16 位定时器/计数器 2 中断 */
    SCU_NMIIRQ_T16N3    = 11,     /* 16 位定时器/计数器 3 中断 */
    SCU_NMIIRQ_T32N0    = 12,     /* 32 位定时器/计数器 0 中断 */
    SCU_NMIIRQ_WDTINT    = 16,     /* 看门狗中断 */
    SCU_NMIIRQ_RTCINT    = 17,     /* 实时时钟中断 */
    SCU_NMIIRQ_KINT      = 18,     /* 外部按键中断 */
    SCU_NMIIRQ_ADCINT    = 19,     /* 模数转换中断 */
    SCU_NMIIRQ_LVDINT    = 21,     /* 低电压检测中断 */
    SCU_NMIIRQ_PLCLKINT  = 22,     /* PLL 失锁中断 */
    SCU_NMIIRQ_UART0     = 23,     /* UART0 中断 */
    SCU_NMIIRQ_UART1     = 24,     /* UART1 中断 */
    SCU_NMIIRQ_EUART0    = 25,     /* EUART0 中断 */
    SCU_NMIIRQ_SPI0INT    = 27,     /* SPI0 中断 */
    SCU_NMIIRQ_SPI1INT    = 28,     /* SPI1 中断 */
    SCU_NMIIRQ_IIC0INT    = 29,     /* I2C0 中断 */
    SCU_NMIIRQ_CCMINT     = 31,     /* 停振检测中断 */
} SCU_TYPE_NMICS;
```

3.5.2 SCU_GetPWRCFlagStatus函数

- ◆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_Flag: PWRC 复位状态寄存器标志位
- ◆返回值: SET/RESET

/* PWRC 复位状态寄存器标志位 */

```
typedef enum {
    SCU_PWRC_PORF      = (1 << 0),      //POR 总复位标志
    SCU_PWRC_PORRCF    = (1 << 1),      //PORRC 复位标志
    SCU_PWRC_PORRSTF   = (1 << 2),      //PORRST 复位标志
    SCU_PWRC_BORF      = (1 << 3),      //BOR 复位标志
    SCU_PWRC_WDTRSTF   = (1 << 4),      //WDT 复位标志
    SCU_PWRC_MRSTF     = (1 << 5),      //MRSTn 丢失标志位
    SCU_PWRC_SOFTRSTF  = (1 << 6),      //软件丢失标志位
    SCU_PWRC_POR_LOST  = (1 << 7),      //POR 丢失标志位
    SCU_PWRC_CFGRST    = (1 << 8),      //配置字读取
} SCU_TYPE_PWRC;
```

3.5.3 SCU_ClearPWRCFlagBit函数

- ◆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_Flag: PWRC 复位状态寄存器标志位, 参见 SCU_TYPE_PWRC 枚举类型
- ◆返回值: 无

3.5.4 SCU_GetLVDFlagStatus函数

- ◆函数原型: FlagStatus SCU_GetLVDFlagStatus(SCU_TYPE_LVDCON LVD_Flag);
- ◆功能描述: 获取 LVDD 寄存器标志位状态
- ◆输入参数:
LVD_Flag: LVDD 状态标志位
- ◆返回值: SET/RESET

```
typedef enum {
    SCU_LVDFlag_IF    = (1 << 8),      //LVD 中断标志
    SCU_LVDFlag_Out    = (1 << 15),    //输出状态位
} SCU_TYPE_LVD0CON;
```

3.5.5 SCU_SysClkSelect函数

- ◆函数原型: void SCU_SysClkSelect(SCU_TYPE_SYSCCLK Sysclk);
- ◆功能描述: 系统时钟选择

◆输入参数:

Sysclk: 时钟类型

◆返回值: 无

```
typedef enum {
    SCU_CLK_HRC   = 0x0 ,           //4/16MHz RC 时钟
    SCU_CLK_LRC   = 0x1 ,           //32KHZ RC 时钟
    SCU_CLK_XTAL  = 0x2 ,           //XTAL 时钟
} SCU_TYPE_SYSClk;
```

3.5.6 SCU_GetSysClk函数

◆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, 参见 SCU_TYPE_SYSClk 枚举类型

3.5.7 DeviceClockAllEnable函数

◆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

3.5.8 DeviceClockAllDisable函数

◆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

3.5.9 SCU_OpenXTAL函数

◆函数原型: void SCU_OpenXTAL(void)

◆功能描述: 开启外部时钟

◆输入参数: 无

◆返回值: 无

3.5.10 DeviceClock_Config函数

◆函数原型: void DeviceClock_Config(SCU_TYPE_Periph
tpe_periph ,TYPE_FUNCEN NewState)

◆功能描述: 配置外设是否使能

◆输入参数:

tpe_periph : 选择的外设类型

NewState : Disable 或 Enable

◆返回值: 无

3.5.11 PLLClock_Config函数

◆函数原型: void PLLClock_Config(TYPE_FUNCEN pll_en , SCU_PLL_Origin
pll_origin ,SCU_PLL_Out pll_out,TYPE_FUNCEN sys_pll)

◆功能描述: PLL 时钟配置,并设置 PLL 时钟为系统时钟

◆输入参数:

pll_en:是否开启 PLL

pll_origin: pll 时钟源选择

pll_out: pll 输出频率选择

sys_pll: 系统时钟是否使用 PLL 时钟

◆返回值: 无

3.6 库函数应用示例

```
int main()
{
    SystemInit();                //系统初始化，配置系统时钟源
    DeviceClockAllEnable();      //打开所有外设时钟

    T16N3Init();                 //Init timer
    KeyInit();                   //Init key
    IIC0_MasterInit();           //Init IIC0
    LEDInit();                   //Init LED
    User_SysTickInit();          //Init system tick
    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

无

4.3 宏定义

无

4.4 库函数

库函数定义于 lib_scs.c 中，声明于 lib_scs.h 中。

4.4.1 NVIC_EnableIRQ函数

- ◆函数原型：void NVIC_EnableIRQ(IRQn_Type IRQn)
- ◆功能描述：使能某个 NVIC 中断
- ◆输入参数：
IRQn: NVIC 中断编号，详见表 4-1
- ◆返回值：无

NVIC 中断编号 IRQn:

枚举元素	数值	描述
PINT0_IRQn	0	外部端口中断 0
PINT1_IRQn	1	外部端口中断 1
PINT2_IRQn	2	外部端口中断 2
PINT3_IRQn	3	外部端口中断 3
PINT4_IRQn	4	外部端口中断 4
PINT5_IRQn	5	外部端口中断 5
PINT6_IRQn	6	外部端口中断 6
PINT7_IRQn	7	外部端口中断 7

枚举元素	数值	描述
T16N0_IRQn	8	16 位定时器/计数器 0 中断
T16N1_IRQn	9	16 位定时器/计数器 1 中断
T16N2_IRQn	10	16 位定时器/计数器 2 中断
T16N3_IRQn	11	16 位定时器/计数器 3 中断
T32N0_IRQn	12	32 位定时器/计数器 0 中断
Reserved0_IRQn	13	保留，不可使用
Reserved1_IRQn	14	保留，不可使用
Reserved2_IRQn	15	保留，不可使用
WDTINT_IRQn	16	看门狗中断
RTCINT_IRQn	17	实时时钟中断
KINT_IRQn	18	外部按键输入中断
ADCINT_IRQn	19	模数转换中断
Reserved3_IRQn	20	保留，不可使用
LVDINT_IRQn	21	低电压检测中断
PLLLKINT_IRQn	22	PLL 失锁中断
UART0_IRQn	23	UART0 中断
UART1_IRQn	24	UART1 中断
EUART0_IRQn	25	EUART0 中断
Reserved4_IRQn	26	保留，不可使用
SPI0INT_IRQn	27	SPI0 中断
SPI1INT_IRQn	28	SPI1 中断
IIC0INT_IRQn	29	I2C0 中断
Reserved5_IRQn	30	保留，不可使用
CCMINT_IRQn	31	停振检测中断

表 4-1 NVIC 中断编号

4.4.2 NVIC_DisableIRQ函数

- ◆函数原型：void NVIC_DisableIRQ(IRQn_Type IRQn)
- ◆功能描述：禁能某个 NVIC 中断
- ◆输入参数：
 - IRQn: NVIC 中断编号，详见表 2-1
- ◆返回值：无

4.4.3 NVIC_SetPriority函数

- ◆函数原型: void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
- ◆功能描述: 设置某个 NVIC 中断的中断优先级
- ◆输入参数:
 - IRQn: NVIC 中断编号, 详见表 2-1
 - priority: 中断优先级, 有效范围是[0, 3]
- ◆返回值: 无

4.4.4 NVIC_GetPriority函数

- ◆函数原型: uint32_t NVIC_GetPriority(IRQn_Type IRQn)
- ◆功能描述: 读取某个 NVIC 中断的中断优先级
- ◆输入参数:
 - IRQn: NVIC 中断编号, 详见表 2-1
- ◆返回值:
 - 0: 中断优先级是 0
 - 1: 中断优先级是 1
 - 2: 中断优先级是 2
 - 3: 中断优先级是 3

4.5 库函数应用示例

```
int main()
{
    SystemInit();           //系统初始化, 配置时钟源
    DeviceClockAllEnable(); //使能外设时钟

    UARTInit();             //Init uart
    NVIC_EnableIRQ(UART0_IRQn); //使能 UART0 中断
    UserFunction();         //用户程序
}
```

第5章 通用输入输出GPIO

5.1 功能概述

- ◆ 支持 2 组 GPIO 端口，最多支持 46 个双向 I/O 端口。
- ◆ 支持 8 路外部端口中断输入，中断触发方式可配置。
- ◆ 支持 8 路外部按键中断输入，中断触发方式可配置。

5.2 寄存器结构

GPIO 寄存器结构定义在 HR8P506.h

typedef struct

```
{  
    __IO GPIO_PAPORT_Typedef PAPORT;  
    __IO GPIO_PADATA_Typedef PADATA;  
    __O GPIO_PADATABSR_Typedef PADATABSR;  
    __O GPIO_PADATABCR_Typedef PADATABCR;  
    __O GPIO_PADATABRR_Typedef PADATABRR;  
    __IO GPIO_PADIR_Typedef PADIR;  
    __O GPIO_PADIRBSR_Typedef PADIRBSR;  
    __O GPIO_PADIRBCR_Typedef PADIRBCR;  
    __O GPIO_PADIRBRR_Typedef PADIRBRR;  
    __IO GPIO_PAFUNC0_Typedef PAFUNC0;  
    __IO GPIO_PAFUNC1_Typedef PAFUNC1;  
    __IO GPIO_PAFUNC2_Typedef PAFUNC2;  
    __IO GPIO_PAFUNC3_Typedef PAFUNC3;  
    __IO GPIO_PAINEB_Typedef PAINEB;  
    __IO GPIO_PAODE_Typedef PAODE;  
    __IO GPIO_PAPUE_Typedef PAPUE;  
    __IO GPIO_PAPDE_Typedef PAPDE;  
    __IO GPIO_PADS_Typedef PADS;  
    uint32_t RESERVED0[14];  
    __IO GPIO_PBPORT_Typedef PBPORT;  
    __IO GPIO_PBDATA_Typedef PBDATA;  
    __O GPIO_PBDATABSR_Typedef PBDATABSR;  
    __O GPIO_PBDATABCR_Typedef PBDATABCR;  
    __O GPIO_PBDATABRR_Typedef PBDATABRR;  
    __IO GPIO_PBDIR_Typedef PBDIR;  
    __O GPIO_PBDIRBSR_Typedef PBDIRBSR;  
    __O GPIO_PBDIRBCR_Typedef PBDIRBCR;  
    __O GPIO_PBDIRBRR_Typedef PBDIRBRR;  
    __IO GPIO_PBFUNC0_Typedef PBFUNC0;  
    __IO GPIO_PBFUNC1_Typedef PBFUNC1;  
    uint32_t RESERVED1[2];  
    __IO GPIO_PBINEB_Typedef PBINEB;
```

```

__IO GPIO_PBODE_Typedef PBODE;
__IO GPIO_PBPUE_Typedef PBPUE;
__IO GPIO_PBPDE_Typedef PBPDE;
__IO GPIO_PBDS_Typedef PBDS;
uint32_t RESERVED2[142];
__IO GPIO_PINTIE_Typedef PINTIE;
__IO GPIO_PINTIF_Typedef PINTIF;
__IO GPIO_PINTSEL_Typedef PINTSEL;
__IO GPIO_PINTCFG_Typedef PINTCFG;
__IO GPIO_KINTIE_Typedef KINTIE;
__IO GPIO_KINTIF_Typedef KINTIF;
__IO GPIO_KINTSEL_Typedef KINTSEL;
__IO GPIO_KINTCFG_Typedef KINTCFG;
uint32_t RESERVED3[4];
__IO GPIO_IOINTFLTS_Typedef IOINTFLTS;
uint32_t RESERVED4[3];
__IO GPIO_TMRFLTSEL_Typedef TMRFLTSEL;
__IO GPIO_SPIFLTSEL_Typedef SPIFLTSEL;
uint32_t RESERVED5[14];
__IO GPIO_TXPWM_Typedef TXPWM;
__IO GPIO_BUZC_Typedef BUZC;
} GPIO_TypeDef;

```

```

#define APB_BASE (0x40000000UL)
#define GPIO_BASE (APB_BASE + 0x00400)
#define GPIO ((GPIO_TypeDef *) GPIO_BASE )

```

5.3 宏定义

无。

5.4 库函数

库函数定义于 lib_gpio.c 中，声明于 lib_gpio.h 中。

5.4.1 GPIO模块的API

5.4.1.1 GPIO_Init函数

- ◆函数原型：void GPIO_Init(GPIO_Pin Pin, GPIO_InitSettingType *InitSet)
- ◆功能描述：引脚的初始化
- ◆输入参数：
 - Pin: 引脚编号，详见表 5-1。
 - InitSet: 初始化配置结构体。
- ◆返回值：无

引脚编号 GPIO_Pin:

枚举元素	数值	描述
GPIO_Pin_B0	0	管脚
GPIO_Pin_B1	1	管脚
GPIO_Pin_B2	2	管脚
GPIO_Pin_B3	3	管脚
GPIO_Pin_B4	4	管脚
GPIO_Pin_B5	5	管脚
GPIO_Pin_B6	6	管脚
GPIO_Pin_B7	7	管脚
GPIO_Pin_B8	8	管脚
GPIO_Pin_B9	9	管脚
GPIO_Pin_B10	10	管脚
GPIO_Pin_B11	11	管脚
GPIO_Pin_B12	12	管脚
GPIO_Pin_B13	13	管脚
GPIO_Pin_A0	14	管脚
GPIO_Pin_A1	15	管脚
GPIO_Pin_A2	16	管脚
GPIO_Pin_A3	17	管脚
GPIO_Pin_A4	18	管脚
GPIO_Pin_A5	19	管脚
GPIO_Pin_A6	20	管脚
GPIO_Pin_A7	21	管脚
GPIO_Pin_A8	22	管脚
GPIO_Pin_A9	23	管脚
GPIO_Pin_A10	24	管脚
GPIO_Pin_A11	25	管脚
GPIO_Pin_A12	26	管脚
GPIO_Pin_A13	27	管脚
GPIO_Pin_A14	28	管脚
GPIO_Pin_A15	29	管脚

枚举元素	数值	描述
GPIO_Pin_A16	30	管脚
GPIO_Pin_A17	31	管脚
GPIO_Pin_A18	32	管脚
GPIO_Pin_A19	33	管脚
GPIO_Pin_A20	34	管脚
GPIO_Pin_A21	35	管脚
GPIO_Pin_A22	36	管脚
GPIO_Pin_A23	37	管脚
GPIO_Pin_A24	38	管脚
GPIO_Pin_A25	39	管脚
GPIO_Pin_A26	40	管脚
GPIO_Pin_A27	41	管脚
GPIO_Pin_A28	42	管脚
GPIO_Pin_A29	43	管脚
GPIO_Pin_A30	44	管脚
GPIO_Pin_A31	45	管脚

表 5-1 GPIO 引脚编号

初始化配置结构体 InitSet:

```
typedef struct {
    GPIO_Pin_Signal Signal; /* 引脚上的信号类型，只有模拟和数字两种 */
    GPIO_Direction Dir; /* 数字引脚时，引脚输入、输出方向选择 */
    GPIO_Reuse_Func Func; /* 数字引脚时，引脚复用功能选择 */
    GPIO_ODE_Output ODE; /* 数字引脚、输出方向时，开漏输出选择 */
    GPIO_DS_Output DS; /* 数字引脚、输出方向时，电流输出大小选择 */
    GPIO_PUE_Input PUE; /* 数字引脚、输入方向时，弱上拉选择 */
    GPIO_PDE_Input PDE; /* 数字引脚、输入方向时，弱下拉选择 */
} GPIO_InitSettingType;
```

引脚信号类型:

```
typedef enum {
    GPIO_Pin_Signal_Digital = 0, // 数字信号引脚
    GPIO_Pin_Signal_Analog = 1, // 模拟信号引脚
} GPIO_Pin_Signal;
```

引脚方向选择:

```
typedef enum {  
    GPIO_Direction_Input = 0,        // 输入方向  
    GPIO_Direction_Output = 1,       // 输出方向  
} GPIO_Direction;
```

引脚复用功能选择:

```
typedef enum {  
    GPIO_Reuse_Func0 = 0,            // 引脚复用功能 0  
    GPIO_Reuse_Func1 = 1,            // 引脚复用功能 1  
    GPIO_Reuse_Func2 = 2,            // 引脚复用功能 2  
    GPIO_Reuse_Func3 = 3,            // 引脚复用功能 3  
} GPIO_Reuse_Func;
```

引脚输出开漏使能位:

```
typedef enum {  
    GPIO_ODE_Output_Disable = 0,     // 开漏禁止  
    GPIO_ODE_Output_Enable = 1,      // 开漏使能  
} GPIO_ODE_Output;
```

引脚输出电流驱动能力选择:

```
typedef enum {  
    GPIO_DS_Output_Normal = 0,       // 普通电流输出  
    GPIO_DS_Output_Strong = 1,       // 强电流输出  
} GPIO_DS_Output;
```

引脚输入弱上拉使能:

```
typedef enum {  
    GPIO_PUE_Input_Disable = 0,      // 弱上拉禁止  
    GPIO_PUE_Input_Enable = 1,       // 弱上拉使能  
} GPIO_PUE_Input;
```

引脚输入弱下拉使能:

```
typedef enum {  
    GPIO_PDE_Input_Disable = 0,      // 弱下拉禁止  
    GPIO_PDE_Input_Enable = 1,       // 弱下拉使能  
} GPIO_PDE_Input;
```

5.4.1.2 GPIO_ReadBit函数

◆函数原型: uint32_t GPIO_ReadBit(GPIO_Pin Pin)

◆功能描述: 读取引脚的电平值

◆输入参数:

Pin: 引脚编号, 详见表 3-1

◆返回值:

0: 引脚的电平值为 0

1: 引脚的电平值为 1

5.4.1.3 GPIO_WriteBit函数

- ◆函数原型: void GPIO_WriteBit(GPIO_Pin Pin, uint32_t value)
- ◆功能描述: 修改引脚的电平值
- ◆输入参数:
 - Pin: 引脚编号, 详见表 3-1
 - value: 引脚电平值, 只有 bit0 有效
- ◆返回值: 无

5.4.1.4 5.4.1.4void GPIO_SetBit函数

- ◆函数原型: void GPIO_SetBit(GPIO_Pin Pin)
- ◆功能描述: 引脚电平置 1
- ◆输入参数:
 - Pin: 引脚编号, 详见表 3-1
- ◆返回值: 无

5.4.1.5 5.4.1.5void GPIO_ResetBit函数

- ◆函数原型: void GPIO_ResetBit(GPIO_Pin Pin)
- ◆功能描述: 引脚电平置 0
- ◆输入参数:
 - Pin: 引脚编号, 详见表 3-1
- ◆返回值: 无

5.4.1.6 5.4.1.5void GPIO_ToggleBit函数

- ◆函数原型: void GPIO_ToggleBit(GPIO_Pin Pin)
- ◆功能描述: 引脚电平取反
- ◆输入参数:
 - Pin: 引脚编号, 详见表 3-1
- ◆返回值: 无

5.4.2 外部端口模块的API

5.4.2.1 PINT_Init函数

- ◆函数原型: void PINT_Init(GPIO_Pin Pin, PINT_InitSettingType *InitSet)
- ◆功能描述: 外部端口初始化
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
 - InitSet: 初始化配置结构体
- ◆返回值: 无

初始化配置结构体 InitSet:

```
typedef struct {
    PINT_IE_Set IE_Set;           // 中断是否使能
    PINT_Trigger_Style Trigger_Style; // 中断触发方式
} PINT_InitSettingType;
```

外部端口中断使能:

```
typedef enum {
    PINT_IE_Set_Disable = 0,           // 中断禁止
    PINT_IE_Set_Enable = 1,           // 中断使能
} PINT_IE_Set;
```

外部端口中断触发方式:

```
typedef enum {
    PINT_Trigger_Rising_Edge = 0,      // 上升沿触发中断
    PINT_Trigger_Trailing_Edge = 1,    // 下降沿触发中断
    PINT_Trigger_High_Level = 2,       // 高电平触发中断
    PINT_Trigger_Low_Level = 3,        // 低电平触发中断
    PINT_Trigger_Both_Edge = 4,        // 上升沿和下降沿都触发中断
} PINT_Trigger_Style;
```

5.4.2.2 PINT_ClearITFlag函数

- ◆函数原型: void PINT_ClearITFlag(GPIO_Pin Pin)
- ◆功能描述: 清除指定引脚引起的外部端口中断标志位
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
- ◆返回值: 无

5.4.2.3 PINT_GetITFlag函数

- ◆函数原型: uint32_t PINT_GetITFlag(GPIO_Pin Pin)
- ◆功能描述: 读取指定引脚引起的外部端口中断标志位
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
- ◆返回值:
 - 0: 中断标志位是 0
 - 1: 中断标志位是 1

5.4.3 外部按键模块的API

5.4.3.1 KINT_Init函数

- ◆函数原型: void KINT_Init(GPIO_Pin Pin, KINT_InitSettingType *InitSet)
- ◆功能描述: 外部按键初始化
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
 - InitSet: 初始化配置结构体
- ◆返回值: 无

初始化配置结构体 InitSet:

```
typedef struct {
    KINT_IE_Set IE_Set;           // 中断是否使能
    KINT_Trigger_Style Trigger_Style; // 中断触发方式
} KINT_InitSettingType;
```

KINT 中断使能:

```
typedef enum {
    KINT_IE_Set_Disable = 0,      // 中断禁止
    KINT_IE_Set_Enable = 1,       // 中断使能
} KINT_IE_Set;
```

KINT 中断方式:

```
typedef enum {
    KINT_Trigger_Rising_Edge = 0, // 上升沿触发中断
    KINT_Trigger_Trailing_Edge = 1, // 下降沿触发中断
    KINT_Trigger_High_Level = 2, // 高电平触发中断
    KINT_Trigger_Low_Level = 3, // 低电平触发中断
    KINT_Trigger_Both_Edge = 4, // 上升沿和下降沿都触发中断
} KINT_Trigger_Style;
```

5.4.3.2 KINT_ClearITFlag函数

- ◆函数原型: void KINT_ClearITFlag(GPIO_Pin Pin)
- ◆功能描述: 清除指定引脚引起的外部按键中断标志位
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
- ◆返回值: 无

5.4.3.3 KINT_GetITFlag函数

- ◆函数原型: uint32_t KINT_GetITFlag(GPIO_Pin Pin)
- ◆功能描述: 读取指定引脚引起的外部按键中断标志位
- ◆输入参数:
 - Pin: 输入引脚编号, 详见表 3-1
- ◆返回值:
 - 0: 外部按键中断标志位是 0
 - 1: 外部按键中断标志位是 1

5.5 库函数应用示例

```
int main(void)
{
    GPIO_InitSettingType x;

    SystemInit();
```

```
DeviceClockAllEnable();
x.Signal = GPIO_Pin_Signal_Digital;
x.Func = GPIO_Reuse_Func0;          /*端口复用使用功能 0 */
x.Dir = GPIO_Direction_Output;     /* 输出方向*/
x.ODE = GPIO_ODE_Output_Disable;   /* 推挽输出 */
x.DS = GPIO_DS_Output_Normal;      /* 普通电流驱动*/
GPIO_Init(GPIO_Pin_A6, &x);
GPIO_Init(GPIO_Pin_A7, &x);
GPIO_Init(GPIO_Pin_A8, &x);
GPIO_Init(GPIO_Pin_A9, &x);
GPIO_WriteBit(GPIO_Pin_A6, 1);
GPIO_WriteBit(GPIO_Pin_A7, 1);
GPIO_WriteBit(GPIO_Pin_A8, 1);
GPIO_WriteBit(GPIO_Pin_A9, 0);     /* 输出低电平，点亮 LED */
}
```

第6章 定时器/计数器T16N/T32N

6.1 功能概述

HR8P506 芯片支持 4 个 16 位定时器 T16N0/1/2/3 及 1 个 32 位定时器 T32N0。

6.1.1 T16N

- ◆ 1 个 8 位可配置预分频器，分频时钟作为 T16N_CNT0/1 定时/计数时钟
 - ◇ 预分频时钟源可选：PCLK 或 T16N0CK0/T16N0CK1
 - ◇ 预分频计数器可由 T16N_PRECNT 寄存器设定预设值
 - ◇ 分频比由寄存器 T16N_PREMAT 设定
- ◆ 2 个 16 位可配置定时/计数寄存器 T16N_CNT0/1
 - ◇ T16N_CNT1 仅在独立调制工作模式或 T16N_CNT1 峰值触发 ADC 转换被使能时可用
- ◆ 2 个 16 位峰值寄存器 T16N_TOP0/1
 - ◇ T16N_CNT0/1 计数值达到峰值时被清零
 - ◇ 在调制工作模式下，若使能 ADC 触发功能，T16N_CNT0/1 计数值达到峰值时可分别产生 ADC 转换触发信号
- ◆ 支持定时/计数工作模式
- ◆ 支持输入捕捉工作模式
- ◆ 支持调制工作模式

6.1.2 T32N

- ◆ 1 个 8 位可配置预分频计数器，所产生分频时钟作为 T32N_CNT 计数器的定时或计数时钟
 - ◇ 预分频时钟源可选：PCLK 或 T32N0CK0/ T32N0CK1
 - ◇ 预分频计数器可由 T32N_PRECNT 寄存器设定计数初值
 - ◇ 分频比由 T32N_PREMAT 寄存器设定
- ◆ 1 个 32 位可配置定时/计数寄存器 T32N_CNT
- ◆ 可配置定时/计数工作模式
 - ◇ 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 HR8P506.h。

typedef struct

```
{
    __IO T16N_CNT0_Typedef CNT0;
    __IO T16N_CNT1_Typedef CNT1;
    __IO T16N_PRECNT_Typedef PRECNT;
    __IO T16N_PREMAT_Typedef PREMAT;
    __IO T16N_CON0_Typedef CON0;
    __IO T16N_CON1_Typedef CON1;
    __IO T16N_CON2_Typedef CON2;
    uint32_t RESERVED0 ;
    __IO T16N_IE_Typedef IE;
    __IO T16N_IF_Typedef IF;
    __IO T16N_PDZ_Typedef PDZ;
    __IO T16N_PTR_Typedef PTR;
    __IO T16N_MAT0_Typedef MAT0;
    __IO T16N_MAT1_Typedef MAT1;
    __IO T16N_MAT2_Typedef MAT2;
    __IO T16N_MAT3_Typedef MAT3;
    __IO T16N_TOP0_Typedef TOP0;
    __IO T16N_TOP1_Typedef TOP1;
} 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 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 )
```

6.3 宏定义

无。

6.4 库函数

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

6.4.1 16 位定时器的API

6.4.1.1 T16Nx_Baselnit函数

◆函数原型：void T16Nx_Baselnit(T16N_TypeDef* T16Nx,T16Nx_BaselnitStruType* T16Nx_BaselnitStruct)

◆功能描述：16 位定时器基本初始化

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

T16Nx_BaselnitStruct: 初始化配置结构体

◆返回值：无

初始化配置结构体：

typedef struct

```
{
    T16Nx_TYPE_CLKS    T16Nx_ClkS;        //时钟源选择
    TYPE_FUNCEN        T16Nx_SYNC;        //外部时钟同步
    T16Nx_TYPE_EDGE    T16Nx_EDGE;        //外部时钟计数边沿选择
    T16Nx_TYPE_MODE    T16Nx_Mode;        //工作模式选择
    unsigned int        T16Nx_PREMAT;      //设置预分频比，范围：1-256
}T16Nx_BaselnitStruType;
```

6. 4. 1. 2 T16Nx_CapInit函数

◆函数原型：void T16Nx_CapInit(T16N_TypeDef* T16Nx,T16Nx_CapInitStruType* T16Nx_CapInitStruct)

◆功能描述：16 位定时器捕捉功能初始化

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

T16Nx_CapInitStruct: 捕捉功能初始化结构体

◆返回值：无

捕捉功能初始化结构体：

typedef struct

```
{
    TYPE_FUNCEN    T16Nx_CAPCAPL1;        //捕捉 1 重载计数器使能
    TYPE_FUNCEN    T16Nx_CAPCAPL0;        //捕捉 0 重载计数器使能
    TYPE_FUNCEN    T16Nx_CapRise;          //上升沿捕捉使能
    TYPE_FUNCEN    T16Nx_CapFall;          //下降沿捕捉使能
    TYPE_FUNCEN    T16Nx_CapIS0;           //输入端口 0 使能
    TYPE_FUNCEN    T16Nx_CapIS1;           //输入端口 1 使能
    T16Nx_TYPE_CAPT T16Nx_CapTime;         //捕捉次数控制
}T16Nx_CapInitStruType;
```

6. 4. 1. 3 T16Nx_PWMOutInit函数

◆函数原型：void T16Nx_PWMOutInit(T16N_TypeDef* T16Nx,T16Nx_PWMInitStruType* T16Nx_PWMInitStruct)

◆功能描述：16 位定时器 PWM 输出初始化

◆输入参数：

T16NIndex: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

T16Nx_PWMInitStruct: PWM 初始化结构体

◆返回值：无。

PWM 初始化结构体：

typedef struct

```
{
    TYPE_FUNCEN    T16Nx_MOE0;              //输出端口 0 使能
    TYPE_FUNCEN    T16Nx_MOE1;              //输出端口 1 使能
    T16Nx_PWMOUT_POLAR_Type T16Nx_POL0;     //T16NxOUT0 输出极性选择位
    T16Nx_PWMOUT_POLAR_Type T16Nx_POL1;     //T16NxOUT1 输出极性选择位
    T16Nx_PWMTYPE_MODE T16Nx_PWMMODE;       //T16PWM 输出模式选择
}
```



```
TYPE_FUNCEN PWMDZE; //PWM 互补模式死区使能
}T16Nx_PWMInitStruType;
```

6. 4. 1. 4 T16Nx_PWMPDZ_Config函数

◆函数原型: void T16Nx_PWMPDZ_Config(T16N_TypeDef* T16Nx,unsigned int PWM_PDZ_data)

◆功能描述: 16 位定时器的 PWM 互补模式死区宽度配置

◆输入参数:

T16NIndex: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

PWM_PDZ_data: 死区周期配置

6. 4. 1. 5 T16Nx_MATxITConfig函数

◆函数原型:

void T16Nx_MAT0ITConfig(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATCON Type)

void T16Nx_MAT1ITConfig(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATCON Type)

void T16Nx_MAT2ITConfig(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATCON Type)

void T16Nx_MAT3ITConfig(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATCON Type)

◆功能描述: T16Nx 匹配后的工作模式配置

◆输入参数:

T16NIndex: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

Type:匹配后的工作模式

匹配寄存器值匹配后的工作模式:

typedef enum

```
{
    T16Nx_Go_No = 0x0, //匹配寄存器值匹配后的工作模式:继续计数, 不产生中断
    T16Nx_Hold_Int = 0x1, //匹配寄存器值匹配后的工作模式:保持计数, 产生中断
    T16Nx_Clr_Int = 0x2, //匹配寄存器值匹配后的工作模式:清零并重新计数, 产生中断
    T16Nx_Go_Int = 0x3, //匹配寄存器值匹配后的工作模式:继续计数, 产生中断
}T16Nx_TYPE_MATCON;
```

6. 4. 1. 6 T16Nx_MATxOutxConfig函数

◆函数原型:

void T16Nx_MAT0Out0Config(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATOUT Type)

void T16Nx_MAT1Out0Config(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATOUT Type)

void T16Nx_MAT2Out1Config(T16N_TypeDef* T16Nx,T16Nx_TYPE_MATOUT Type)

void T16Nx_MAT3Out1Config(T16N_TypeDef* T16Nx,T16Nx_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:

T16NIndex: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

Type: 匹配后端口的工作模式

匹配寄存器值匹配后输出端口的工作模式:

typedef enum

```
{
    T16Nx_Out_Hold = 0x0, //匹配寄存器值匹配后输出端口的工作模式: 保持
```

```
T16Nx_Out_Low = 0x1 ,      //匹配寄存器值匹配后输出端口的工作模式：清 0  
T16Nx_Out_High = 0x2 ,    //匹配寄存器值匹配后输出端口的工作模式：置 1  
T16Nx_Out_Switch = 0x3 ,  //匹配寄存器值匹配后输出端口的工作模式：取反  
}T16Nx_TYPE_MATOUT;
```

6. 4. 1. 7 T16Nx_ITConfig函数

◆函数原型：void T16Nx_ITConfig(T16N_TypeDef* T16Nx,T16Nx_TYPE_IT Type,TYPE_FUNCEN NewState)

◆功能描述：T16N 中断配置

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Type: 中断类型

NewState: 使能/失能

6. 4. 1. 8 T16Nx_SetCNT0 函数

◆函数原型：void T16Nx_SetCNT0(T16N_TypeDef* T16Nx,uint16_t Value)

◆功能描述：设置 CNT0 计数值

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 16 位数值

6. 4. 1. 9 T16Nx_SetCNT1 函数

◆函数原型：void T16Nx_SetCNT1(T16N_TypeDef* T16Nx,uint16_t Value)

◆功能描述：设置 CNT1 计数值

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 16 位数值

6. 4. 1. 10 T16Nx_SetPREMAT函数

◆函数原型：void T16Nx_SetPREMAT(T16N_TypeDef* T16Nx,uint8_t Value)

◆功能描述：设置预分频比

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 预分频比

6. 4. 1. 11 T16Nx_SetPRECNT函数

◆函数原型：void T16Nx_SetPRECNT(T16N_TypeDef* T16Nx,uint8_t Value)

◆功能描述：设置预分频计数器

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 预分频计数器

6. 4. 1. 12 T16Nx_SetMATx函数

◆函数原型：

void T16Nx_SetMAT0(T16N_TypeDef* T16Nx,uint16_t Value)

void T16Nx_SetMAT1(T16N_TypeDef* T16Nx,uint16_t Value)

```
void T16Nx_SetMAT2(T16N_TypeDef* T16Nx,uint16_t Value)
```

```
void T16Nx_SetMAT3(T16N_TypeDef* T16Nx,uint16_t Value)
```

◆功能描述：设置匹配寄存器

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 匹配寄存器

6. 4. 1. 13 T16Nx_SetTOPx函数

◆函数原型：

```
void T16Nx_SetTOP0(T16N_TypeDef* T16Nx,uint16_t Value)
```

```
void T16Nx_SetTOP1(T16N_TypeDef* T16Nx,uint16_t Value)
```

◆功能描述：设置峰值寄存器

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

Value: 峰值寄存器

6. 4. 1. 14 T16Nx_GetMATx函数

◆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: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

◆输出参数：

MATx 的值

6. 4. 1. 15 T16Nx_GetTOPx函数

◆函数原型：

```
uint16_t T16Nx_GetTOP0(T16N_TypeDef* T16Nx)
```

```
uint16_t T16Nx_GetTOP1(T16N_TypeDef* T16Nx)
```

◆功能描述：读取峰值寄存器值

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

◆输出参数：

TOPx 的值

6. 4. 1. 16 T16Nx_GetCNTx函数

◆函数原型：

```
uint16_t T16Nx_GetCNT0(T16N_TypeDef* T16Nx)
```

```
uint16_t T16Nx_GetCNT1(T16N_TypeDef* T16Nx)
```

◆功能描述：读取计数寄存器值

◆输入参数：

T16Nx: 16 位定时器索引，有效值是 T16N0/T16N1/T16N2/T16N3

◆输出参数：

CNTx 的值

6. 4. 1. 17 T16Nx_GetFlagStatus函数

◆函数原型:

FlagStatus T16Nx_GetFlagStatus(T16N_TypeDef* T16Nx, T16Nx_TYPE_IT T16Nx_Flag)

◆功能描述: 读取指定标志位

◆输入参数:

T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

T16Nx_Flag: 中断标志位

◆输出参数:

RESET/SET

6. 4. 1. 18 T16Nx_ClearITPendingBit函数

◆函数原型:

void T16Nx_ClearITPendingBit(T16N_TypeDef* T16Nx, T16Nx_TYPE_IT T16Nx_Flag)

◆功能描述: 清除指定的中断标志位

◆输入参数:

T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

T16Nx_Flag: 中断标志位

6. 4. 1. 19 T16Nx_Enable函数

◆函数原型: void T16Nx_Enable(T16N_TypeDef* T16Nx)

◆功能描述: 使能 T16Nx

◆输入参数:

T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

6. 4. 1. 20 T16Nx_Disable函数

◆函数原型: void T16Nx_Disable(T16N_TypeDef* T16Nx)

◆功能描述: 使能 T16Nx

◆输入参数:

T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

6. 4. 1. 21 T16Nx_PWMBK_Config函数

◆函数原型: void T16Nx_PWMBK_Config(T16N_TypeDef* T16Nx, T16Nx_PWMBK_Type* type)

◆功能描述: 配置 PWM 刹车功能

◆输入参数:

T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

type: 配置 PWM 刹车结构体

PWM 输出刹车控制:

typedef struct

```
{  
    TYPE_FUNCEN T16Nx_PWMBK_EN0;           //PWM 通段 0 刹车使能  
    TYPE_FUNCEN T16Nx_PWMBK_EN1;           //PWM 通段 0 刹车使能
```

```
T16Nx_PWMBKOUT_LEVEL T16Nx_PWMBK_L0; //PWM 通道 0 刹车输出电平选择
T16Nx_PWMBKOUT_LEVEL T16Nx_PWMBK_L1; //PWM 通道 1 刹车输出电平选择
T16Nx_PWMBKP_LEVEL T16Nx_PWMBK_P0; //PWM 通道 0 刹车信号极性选择
T16Nx_PWMBKP_LEVEL T16Nx_PWMBK_P1; //PWM 通道 1 刹车信号极性选择
}T16Nx_PWMBK_Type;
```

6. 4. 1. 22 T16Nx_GetPWMBKF函数

- ◆函数原型: FlagStatus T16Nx_GetPWMBKF(T16N_TypeDef* T16Nx)
- ◆功能描述: 获取 PWMBKF 刹车标志位
- ◆输入参数:
 - T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3
- ◆输出参数:
 - PWMBKF 标志位的值。SET: 发生刹车事件, RESET: 未发生刹车事件

6. 4. 1. 23 T16Nx_ResetPWMBKF函数

- ◆函数原型: void T16Nx_ResetPWMBKF(T16N_TypeDef* T16Nx)
- ◆功能描述: 清除 PWMBKF 标志, 标志清除后 PWM 端口恢复正常
- ◆输入参数:
 - T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3

6. 4. 1. 24 T16Nx_PTR_Config函数

- ◆函数原型: void T16Nx_PTR_Config(T16N_TypeDef* T16Nx, T16Nx_PWMTRE_type Type, TYPE_FUNCEN NewState)
- ◆功能描述: 配置 PWM 调试模式 ADC 触发使能
- ◆输入参数:
 - T16Nx: 16 位定时器索引, 有效值是 T16N0/T16N1/T16N2/T16N3
 - Type: 配置 PWM 调试模式 ADC 触发机构体
 - NewState: Enable/Disable

PWM 调制模式 ADC 触发使能控制机构体:

```
typedef enum
{
    T16Nx_P0MAT0 = 0x02, //PWM 通道 0 匹配 0 触发使能
    T16Nx_P0MAT1 = 0x04, //PWM 通道 0 匹配 1 触发使能
    T16Nx_P0TOP0 = 0x08, //PWM 通道 0 峰值 0 触发使能
    T16Nx_P1MAT2 = 0x20, //PWM 通道 1 匹配 2 触发使能
    T16Nx_P1MAT3 = 0x40, //PWM 通道 1 匹配 3 触发使能
    T16Nx_P1TOP1 = 0x80, //PWM 通道 1 峰值 1 触发使能
}T16Nx_PWMTRE_type;
```

6. 4. 2 32 位定时器的API

6. 4. 2. 1 T32Nx_Baselnit函数

- ◆函数原型: void T32Nx_Baselnit(T32N_TypeDef* T32Nx, T32Nx_BaselnitStruType* T32Nx_BaselnitStruct)

- ◆功能描述: T32Nx 基本初始化
- ◆输入参数:
 - T32Nx: 32 位定时器索引, 有效值是 T32N0
 - T32Nx_BaselnitStruct: 初始化配置结构体
- ◆返回值: 无

初始化配置结构体:

```
typedef struct
{
    T32Nx_TYPE_CLKS    T32Nx_ClkS;        //时钟源选择
    TYPE_FUNCEN        T32Nx_SYNC;        //外部时钟同步
    T32Nx_TYPE_EDGE    T32Nx_EDGE;        //外部时钟计数边沿选择
    T32Nx_TYPE_MODE    T32Nx_Mode;        //工作模式选择
    unsigned int        T32Nx_PREMAT;      //设置预分频比, 范围 : 1-256
}T32Nx_BaselnitStruType;
```

6. 4. 2. 2 T32Nx_CapInit函数

- ◆函数原型: void T32Nx_CapInit(T32N_TypeDef* T32Nx,T32Nx_CapInitStruType* T32Nx_CapInitStruct)
- ◆功能描述: T32Nx 捕捉初始化
- ◆输入参数:
 - T32Nx: 32 位定时器索引, 有效值是 T32N0
 - T32Nx_CapInitStruct: 初始化配置结构体地址
- ◆返回值: 无

捕捉功能初始化结构体定义:

```
typedef struct
{
    TYPE_FUNCEN    T32Nx_CAPCAPL1;    //捕捉 1 重载计数器使能
    TYPE_FUNCEN    T32Nx_CAPCAPL0;    //捕捉 0 重载计数器使能
    TYPE_FUNCEN    T32Nx_CapRise;      //上升沿捕捉使能
    TYPE_FUNCEN    T32Nx_CapFall;      //下降沿捕捉使能
    TYPE_FUNCEN    T32Nx_CapIS0;      //输入端口 0 使能
    TYPE_FUNCEN    T32Nx_CapIS1;      //输入端口 1 使能
    T32Nx_TYPE_CAPT    T32Nx_CapTime;  //捕捉次数控制
}T32Nx_CapInitStruType;
```

6. 4. 2. 3 T32Nx_PMWOutInit函数

- ◆函数原型: void T32Nx_PMWOutInit(T32N_TypeDef* T32Nx,T32Nx_PWMInitStruType* T32Nx_PWMInitStruct)
- ◆功能描述: T32Nx 捕捉初始化
- ◆输入参数:
 - T32Nx: 32 位定时器索引, 有效值是 T32N0

T32Nx_PWMInitStruct: 初始化配置结构体地址

◆返回值: 无

6. 4. 2. 4 T32Nx_MATxITConfig函数

◆函数原型:

void T32Nx_MAT0ITConfig(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATCON Type)

void T32Nx_MAT1ITConfig(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATCON Type)

void T32Nx_MAT2ITConfig(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATCON Type)

void T32Nx_MAT3ITConfig(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATCON Type)

◆功能描述: T32Nx 匹配后的工作模式配置

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Type: 匹配后的工作模式

◆返回值: 无

6. 4. 2. 5 T32Nx_MATxOut0Config函数

◆函数原型:

void T32Nx_MAT0Out0Config(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATOUT Type)

void T32Nx_MAT1Out0Config(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATOUT Type)

void T32Nx_MAT2Out1Config(T32N_TypeDef* T32Nx,T32Nx_TYPE_MATOUT Type)

void T32Nx_MAT3Out1Config(T32N_TypeDef* T32Nx,T32Nx_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:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Type: 匹配后端口的工作模式

◆返回值: 无

6. 4. 2. 6 T32Nx_ITConfig函数

◆函数原型:

void T32Nx_ITConfig(T32N_TypeDef* T32Nx,T32Nx_TYPE_IT Type,TYPE_FUNCEN NewState)

◆功能描述: T32N 中断配置

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Type: 中断类型

NewState: 使能/失能

◆返回值: 无

6. 4. 2. 7 T32Nx_SetCNT函数

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

◆功能描述: 设置 CNT 计数值

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Value: 32 位数值

◆返回值: 无

6. 4. 2. 8 T32Nx_SetPREMAT函数

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

◆功能描述: 设置预分频比

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Value: 1-256

◆返回值: 无

6. 4. 2. 9 T32Nx_SetPRECNT函数

◆函数原型: void T32Nx_SetPRECNT(T32N_TypeDef* T32Nx,uint8_t Value)

◆功能描述: 设置预分频计数器

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Value: 1-256

◆返回值: 无

6. 4. 2. 10 T32Nx_SetMATx函数

◆函数原型:

void T32Nx_SetMAT0(T32N_TypeDef* T32Nx,uint32_t Value)

void T32Nx_SetMAT1(T32N_TypeDef* T32Nx,uint32_t Value)

void T32Nx_SetMAT2(T32N_TypeDef* T32Nx,uint32_t Value)

void T32Nx_SetMAT3(T32N_TypeDef* T32Nx,uint32_t Value)

◆功能描述: 设置匹配寄存器

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

Value: 32 位数值

◆返回值: 无

6. 4. 2. 11 T32Nx_GetMATx函数

◆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: 32 位定时器索引, 有效值是 T32N0

◆返回值: 32 位数值

6. 4. 2. 12 T32Nx_GetCNT函数

◆函数原型: uint16_t T32Nx_GetCNT(T32N_TypeDef* T32Nx)

◆功能描述: 读取计数寄存器值

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

◆返回值: 32 位数值

6. 4. 2. 13 T32Nx_GetFlagStatus函数

◆函数原型: FlagStatus T32Nx_GetFlagStatus(T32N_TypeDef* T32Nx,T32Nx_TYPE_IT T32Nx_Flag)

◆功能描述: 读取指定标志位

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

T32Nx_Flag: 中断标志位

◆返回值: RESET/SET

6. 4. 2. 14 T32Nx_ClearITPendingBit函数

◆函数原型: void T32Nx_ClearITPendingBit(T32N_TypeDef* T32Nx,T32Nx_TYPE_IT TIM_Flag)

◆功能描述: 清除指定的中断标志位

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

TIM_Flag: 中断标志位

◆返回值: 无

6. 4. 2. 15 T32Nx_Enable函数

◆函数原型: void T32Nx_Enable(T32N_TypeDef* T32Nx)

◆功能描述: 使能 T32Nx

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

◆返回值: 无

6. 4. 2. 16 T32Nx_Disable函数

◆函数原型: void T32Nx_Disable (T32N_TypeDef* T32Nx)

◆功能描述: 禁止 T32Nx

◆输入参数:

T32Nx: 32 位定时器索引, 有效值是 T32N0

◆返回值: 无

6. 5 库函数应用示例

6. 5. 1 定时器

```
int main(void)
{
    T16Nx_BaseInitStruType x;
    GPIO_InitSettingType y;

    SystemInit();
    DeviceClockAllEnable();

    /* 设置 LED, 使得超时后, 点亮 LED */
    y.Func = GPIO_Reuse_Func0;
```

```

y.Dir = GPIO_Direction_Output;
y.ODE = GPIO_ODE_Output_Disable;
y.DS = GPIO_DS_Output_Normal;
y.PUE = GPIO_PUE_Input_Enable;
y.PDE = GPIO_PDE_Input_Disable;
y.Signal = GPIO_Pin_Signal_Digital;
GPIO_Init(GPIO_Pin_A6, &y);
GPIO_Init(GPIO_Pin_A7, &y);
GPIO_Init(GPIO_Pin_A8, &y);
GPIO_Init(GPIO_Pin_A9, &y);
GPIO_WriteBit(GPIO_Pin_A6, 1);
GPIO_WriteBit(GPIO_Pin_A7, 1);
GPIO_WriteBit(GPIO_Pin_A8, 1);
GPIO_WriteBit(GPIO_Pin_A9, 1);

/* 初始化 T16Nx 定时器*/
x.T16Nx_ClkS = T16Nx_ClkS_PCLK;           //内部时钟 PCLK
x.T16Nx_EDGE = T16Nx_EDGE_Rise;
x.T16Nx_Mode = T16Nx_Mode_TC0;           //定时器计数器模式
x.T16Nx_PREMAT = 1;
x.T16Nx_SYNC = Disable;
T16Nx_BaseInit(T16N0,&x);                 //初始化定时器 T16N0
T16Nx_BaseInit(T16N1,&x);                 //初始化定时器 T16N1
T16Nx_BaseInit(T16N2,&x);                 //初始化定时器 T16N2
T16Nx_BaseInit(T16N3,&x);                 //初始化定时器 T16N3

/* 设置定时器，每隔 40000 个系统时钟产生一次中断 */
T16Nx_MAT0ITConfig(T16N0,T16Nx_Clr_Int); //MAT0 匹配设置
T16Nx_MAT0ITConfig(T16N1,T16Nx_Clr_Int); //MAT0 匹配设置
T16Nx_MAT0ITConfig(T16N2,T16Nx_Clr_Int); //MAT0 匹配设置
T16Nx_MAT0ITConfig(T16N3,T16Nx_Clr_Int); //MAT0 匹配设置
T16Nx_SetCNT0(T16N0,0);                  //设置 T16N0 初始值为 0
T16Nx_SetCNT0(T16N1,0);                  //设置 T16N1 初始值为 0
T16Nx_SetCNT0(T16N2,0);                  //设置 T16N2 初始值为 0
T16Nx_SetCNT0(T16N3,0);                  //设置 T16N3 初始值为 0

T16Nx_SetMAT0(T16N0,40000);              //设置 T16N0MAT0 值为 40000
T16Nx_SetMAT0(T16N1,40000);              //设置 T16N1MAT0 值为 40000
T16Nx_SetMAT0(T16N2,40000);              //设置 T16N2MAT0 值为 40000
T16Nx_SetMAT0(T16N3,40000);              //设置 T16N3MAT0 值为 40000

T16Nx_ITConfig(T16N0,T16Nx_IT_MAT0,Enable);//使能 T16N0 匹配 0 中断
T16Nx_ITConfig(T16N1,T16Nx_IT_MAT0,Enable);//使能 T16N1 匹配 0 中断
T16Nx_ITConfig(T16N2,T16Nx_IT_MAT0,Enable);//使能 T16N2 匹配 0 中断

```

```
T16Nx_ITConfig(T16N3,T16Nx_IT_MAT0,Enable);//使能 T16N3 匹配 0 中断
```

```
NVIC_Init(NVIC_T16N0_IRQn,NVIC_Priority_1,Enable);//中断设置
NVIC_Init(NVIC_T16N1_IRQn,NVIC_Priority_1,Enable);//中断设置
NVIC_Init(NVIC_T16N2_IRQn,NVIC_Priority_1,Enable);//中断设置
NVIC_Init(NVIC_T16N3_IRQn,NVIC_Priority_1,Enable);//中断设置
```

```
T16Nx_Enable(T16N0);
T16Nx_Enable(T16N1);
T16Nx_Enable(T16N2);
T16Nx_Enable(T16N3);
```

```
while (1);
```

```
}
```

6.5.2 计数器

```
int main(void)
```

```
{
```

```
    T16Nx_BaseInitStruType x;
    GPIO_InitSettingType y;
```

```
    SystemInit();
    DeviceClockAllEnable();
```

```
    /* 初始化 T16N3 定时器*/
```

```
    x.T16Nx_ClkS = T16Nx_ClkS_CK0;    //外部时钟 CK0
    x.T16Nx_EDGE = T16Nx_EDGE_Fall;
    x.T16Nx_Mode = T16Nx_Mode_TC0;    //定时器计数器模式
    x.T16Nx_PREMAT = 1;
    x.T16Nx_SYNC = Disable;
    T16Nx_BaseInit(T16N3,&x);        //初始化定时器 T16N3
```

```
    /*外部时钟 PB0 初始化*/
```

```
    y.Func = GPIO_Reuse_Func3;
    y.Dir = GPIO_Direction_Input;
    y.ODE = GPIO_ODE_Output_Disable;
    y.DS = GPIO_DS_Output_Normal;
    y.PUE = GPIO_PUE_Input_Disable;
    y.PDE = GPIO_PDE_Input_Disable;
    y.Signal = GPIO_Pin_Signal_Digital;
    GPIO_Init(GPIO_Pin_B0, &y);
```

```
    y.Func = GPIO_Reuse_Func0;
    y.Dir = GPIO_Direction_Output;
```

```

y.ODE = GPIO_ODE_Output_Disable;
y.DS = GPIO_DS_Output_Normal;
y.PUE = GPIO_PUE_Input_Enable;
y.PDE = GPIO_PDE_Input_Disable;
y.Signal = GPIO_Pin_Signal_Digital;
GPIO_Init(GPIO_Pin_A2, &y);
    GPIO_WriteBit(GPIO_Pin_A2, 1);          /* PA2 输出高电平，使得按键 K4 按下时，产生高电平到 KR2 */

GPIO_Init(GPIO_Pin_B1, &y);
GPIO_WriteBit(GPIO_Pin_B1, 0);          /* PB1 输出低电平，使得按键 K3 按下时，产生低电平到 KR2 */

/* 为 LED 显示做引脚初始化 */
GPIO_Init(GPIO_Pin_A6, &y);
GPIO_Init(GPIO_Pin_A7, &y);
GPIO_Init(GPIO_Pin_A8, &y);
GPIO_Init(GPIO_Pin_A9, &y);
GPIO_WriteBit(GPIO_Pin_A6, 1);
GPIO_WriteBit(GPIO_Pin_A7, 1);
GPIO_WriteBit(GPIO_Pin_A8, 1);
GPIO_WriteBit(GPIO_Pin_A9, 1);

/* 设置计数器，交替按下 K4 和 K3，就会在 PB0 引脚产生下降沿 */
T16Nx_MAT0ITConfig(T16N3,T16Nx_Clr_Int);    //MAT0 匹配设置
T16Nx_SetCNT0(T16N3,0);                    //设置 T16N3 初始值为 0
T16Nx_SetMAT0(T16N3,1);                    //设置 T16N3MAT0 值为 1

T16Nx_ITConfig(T16N3,T16Nx_IT_MAT0,Enable); //使能 T16N3 匹配 0 中断

NVIC_Init(NVIC_T16N3_IRQn,NVIC_Priority_1,Enable);//中断设置

T16Nx_Enable(T16N3);
while (1);
}

```

第7章 数模转换ADC

7.1 功能概述

- ◆ 支持 12 位转换结果，有效精度为 11 位
- ◆ 采样速率最高支持 125ksps (kilo-samples per second)
- ◆ 支持 16 个模拟输入端
- ◆ 支持 ADC 中断，可唤醒睡眠模式（仅在时钟源为 LRC 时唤醒）
- ◆ 支持正负向参考电压可配置
- ◆ 支持转换时钟可配置
- ◆ 支持自动转换比较功能

7.2 寄存器结构

ADC 寄存器结构定义在 HR8P506.h 文件，支持 1 个 ADC。

typedef struct

```
{  
    __IO 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[4];  
    __IO ADC_ACPC_Typedef ACPC;  
    uint32_t RESERVED1;  
    __IO ADC_ACPCMP_Typedef ACPCMP;  
    __IO ADC_ACPMEAN_Typedef ACPMEAN;  
    uint32_t RESERVED2[2];  
    __IO ADC_VREFCON_Typedef VREFCON;  
} 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。

```
#define ADC_ACP_Enable() (ADC->CON0.ACP_EN = 1)  
#define ADC_ACP_Disable() (ADC->CON0.ACP_EN = 0)
```

```
#define ADC_IE_Enable() (ADC->IE.IE = 1)  
#define ADC_IE_Disable() (ADC->IE.IE = 0)
```

```
#define ADC_ACPMINIE_Enable() (ADC->IE.ACPMINIE = 1)
```

```
#define ADC_ACPMINIE_Disable() (ADC ->IE.ACPMINIE = 0)
```

```
#define ADC_ACPMAXIE_Enable() (ADC ->IE.ACPMAXIE = 1)
#define ADC_ACPMAXIE_Disable() (ADC ->IE.ACPMAXIE = 0)
```

```
#define ADC_ACPOVIE_Enable() (ADC ->IE.ACPOVIE = 1)
#define ADC_ACPOVIE_Disable() (ADC ->IE.ACPOVIE = 0)
```

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 ADC_Init函数

◆函数原型：ErrorStatus ADC_Init(ADC_InitStruType * ADC_InitStruct)

◆功能描述：ADC 功能初始化函数

◆输入参数：

ADC_InitStruct :adc 初始化结构体

◆返回值：SUCCESS/ERROR

```
typedef struct {
    ADC_TYPE_CLKS CLKS;           /* ADC 时钟选择 */
    ADC_TYPE_CLKDIV CLKDIV;       /* ADC 预分频 */
    ADC_TYPE_VREF_SEL VREF_SEL;   /* 参考电压选择 */
    ADC_TYPE_VREFP VREFP;         /* 参考电压类型选择 */
    ADC_TYPE_VREFN VREFN;
    ADC_TYPE_CHS CHS;             /* 模拟通道选择 */
    ADC_TYPE_SMPS SMPS;           /* 控制模式选择 */
    uint8_t ST; /* 硬件采样时间 (当控制模式为硬件时设置) 0~31 */
    ADC_TYPE_VDD5_FLAG_EN VDD5_EN; /* VDD 使能选择 */
}ADC_InitStruType;
/* A/D 时钟源选择 */
typedef enum {
    ADC_CLKS_PCLK = 0,           /* PCLK */
    ADC_CLKS_LRC = 1,           /* LRC(32KHz) */
}ADC_TYPE_CLKS;

/* A/D 时钟源预分频选择 */
typedef enum {
    ADC_CLKDIV_1_1 = 0,         /* 1:1 */
    ADC_CLKDIV_1_2 = 1,         /* 1:2 */
    ADC_CLKDIV_1_4 = 2,         /* 1:4 */
    ADC_CLKDIV_1_8 = 3,         /* 1:8 */
    ADC_CLKDIV_1_16 = 4,        /* 1:16 */
    ADC_CLKDIV_1_32 = 5,        /* 1:32 */
    ADC_CLKDIV_1_64 = 6,        /* 1:64 */
}
```

```

        ADC_CLKDIV_1_256 = 7,                /* 1:256 */
    }ADC_TYPE_CLKDIV;

/* 内部参考电压选择 */
typedef enum {
    ADC_VREF_SEL_1_8 = 0,                    /* 内部参考电压 1.8v*/
    ADC_VREF_SEL_2_6 = 1,                    /* 内部参考电压 2.6v*/
}ADC_TYPE_VREF_SEL;

/* A/D 正向参考电压选择 */
typedef enum {
    ADC_VREFP_VDD = 0,                       /* 选择 AD 的工作电压 */
    ADC_VREFP_IO = 1,                        /* 选择内部参考电压 AVREFP,端口复用作普通 I/O 端口*/
    ADC_VREFP_VREF = 2,                      /* 选择内部参考电压 AVREFP,端口输出内部参考电压
VREF*/
    ADC_VREFP_AVREFP = 3                     /* 外部参考电压 AVREFP,电压不能高于 AD 的工作电压 */
}ADC_TYPE_VREFP;

/* A/D 负向参考电压选择*/
typedef enum {
    ADC_VREFN_VSS = 0,                       /* 内部地电压 VSS */
    ADC_VREFN_AVREFN = 1,                    /* 外部参考电压 AVREFN */
    ADC_VREFN_MAX = 2,
}ADC_TYPE_VREFN;

/* A/D 模拟通道选择位 */
typedef enum {
    ADC_CHS_AIN0 = 0,                        /* 通道 0 */
    ADC_CHS_AIN1 = 1,                        /* 通道 1 */
    ADC_CHS_AIN2 = 2,                        /* 通道 2 */
    ADC_CHS_AIN3 = 3,                        /* 通道 3 */
    ADC_CHS_AIN4 = 4,                        /* 通道 4 */
    ADC_CHS_AIN5 = 5,                        /* 通道 5 */
    ADC_CHS_AIN6 = 6,                        /* 通道 6 */
    ADC_CHS_AIN7 = 7,                        /* 通道 7 */
    ADC_CHS_AIN8 = 8,                        /* 通道 8 */
    ADC_CHS_AIN9 = 9,                        /* 通道 9 */
    ADC_CHS_AIN10 = 10,                      /* 通道 10 */
    ADC_CHS_AIN11 = 11,                      /* 通道 11 */
    ADC_CHS_AIN12 = 12,                      /* 通道 12 */
    ADC_CHS_AIN13 = 13,                      /* 通道 13 */
    ADC_CHS_AIN14 = 14,                      /* 通道 14 */
    ADC_CHS_AIN15 = 15,                      /* 通道 15 */
    ADC_CHS_OC = 16,                         /* 温度传感通道*/

```

```
}ADC_TYPE_CHS;
```

```
/* A/D 采样模式选择*/
```

```
typedef enum {
    ADC_SMPS_SOFT = 0,      /* 软件控制 */
    ADC_SMPS_HARD = 1,      /* 硬件控制 */
}ADC_TYPE_SMPS;
```

当设置成软件控制时，ADC 采样只能通过 ADC_SampStart 开始，并且只能通过 ADC_SampStop 结束

```
ADC_TYPE_VDD5_FLAG_EN {ENABLE;DISABLE};
ADC_TYPE_INTR_EN { ENABLE;DISABLE}
```

7.4.2 ADC_ACPConfig函数

◆函数原型：ErrorStatus ADC_ACPConfig(ADC_ACP_TypeDef *ADC_ACP_InitStruct)

◆功能描述：ADC 自动比较功能初始化

◆输入参数：

ADC_ACP_InitStruct 自动比较功能初始化结构体

◆返回值：SUCCESS/ERROR

```
/* 自动比较功能初始化结构体*/
```

```
typedef struct {
    ADC_TYPE_ACP_EN ACP_EN;                /* 自动比较功能使能位*/
    uint16_t ACP_OVER_TIME;                /* 单次自动比较溢出时间(即使不想设置请设置成 0) 0~0x9c3 */
    ADC_TYPE_ACP_TIMES ACP_TIMES;          /* 单次自动比较次数(优先级高于溢出时间) */
    uint16_t ACP_MIN_TARGET;               /* 平均值最低阈值 (设置 0xffff 关闭) 0~0xffff */
    uint16_t ACP_MAX_TARGET;               /* 平均值最高阈值 (设置 0x0 关闭) 0~0xffff */
}ADC_ACP_TypeDef;
```

```
ADC_TYPE_ACP_EN {ENABLE;DISABLE}
```

```
/* 自动转换比较次数 */
```

```
typedef enum {
    ADC_ACP_TIMES_1 = 0,                  /* 1 次 */
    ADC_ACP_TIMES_2 = 1,                  /* 2 次 */
    ADC_ACP_TIMES_4 = 2,                  /* 4 次 */
    ADC_ACP_TIMES_8 = 3,                  /* 8 次 */
}ADC_TYPE_ACP_TIMES;
```

该功能与 ADC_Init 一起进行设置，启动该功能的前提条件是 ADC_Init 中 CONTROL_MODE = ADC_SMPS_HARD；设置完成自动使能 ACP_EN。

当开启 ADC_Start 后，芯片 ADC 开始自动采样并进行比较，直到将 ACP_EN 禁止，芯片 ADC 停止自动采样比较。此时 ADC 获得的平均采样值被清空。

7.4.3 ADC_Start函数

- ◆函数原型：ErrorStatus ADC_Start(void)
- ◆功能描述：ADC 启动函数
- ◆输入参数：无
- ◆返回值：SUCCESS/ERROR

7.4.4 ADC_SoftStart函数

- ◆函数原型：ErrorStatus ADC_SoftStart(void)
- ◆功能描述：ADC 采样软件控制-启动函数
- ◆输入参数：无
- ◆返回值：SUCCESS/ERROR

7.4.5 ADC_SoftStop函数

- ◆函数原型：ErrorStatus ADC_SoftStop(void)
- ◆功能描述：ADC 采样软件控制-停止函数
- ◆输入参数：无
- ◆返回值：SUCCESS/ERROR

7.4.6 ADC_GetConvValue函数

- ◆函数原型：uint16_t ADC_GetConvValue(void)
- ◆功能描述：ADC 获得采样数据函数
- ◆输入参数：无
- ◆返回值：采样数据（12bit）

7.4.7 ADC_GetConvStatus函数

- ◆函数原型：FlagStatus ADC_GetConvStatus(void)
- ◆功能描述：ADC 获得此时状态
- ◆输入参数：无
- ◆返回值：SET/RESET

7.4.8 ADC_GetACPMeanValue函数

- ◆函数原型：uint16_t ADC_GetACPMeanValue(void)
- ◆功能描述：ADC 获得单次自动比较平均值
- ◆输入参数：无
- ◆返回值：单次自动比较平均值

7.4.9 ADC_GetIFStatus函数

◆函数原型: ITStatus ADC_GetIFStatus(ADC_TYPE_IF IFName)

◆功能描述: ADC 获得特定类型中断情况

◆输入参数: IFName 中断类型选择

ADC_IF ADC 中断

ADC_IF_ACPMIN 自动转换低阈值超出中断

ADC_IF_ACPMAX 自动转换高阈值超出中断

ADC_IF_ACPOVER 自动转换溢出中断

◆返回值: SET/RESET

```
typedef enum {  
    ADC_IF                  = 0x01,  
    ADC_IF_ACPMIN = 0x02,  
    ADC_IF_ACPMAX = 0x04,  
    ADC_IF_ACPOVER = 0x08,  
}ADC_TYPE_IF;
```

7.4.10 ADC_ClearIFStatus函数

◆函数原型: ErrorStatus ADC_ClearIFStatus(ADC_TYPE_IF IFName)

◆功能描述: ADC 清除特定类型中断

◆输入参数: IFName 中断类型选择

ADC_IF ADC 中断

ADC_IF_ACPMIN 自动转换低阈值超出中断

ADC_IF_ACPMAX 自动转换高阈值超出中断

ADC_IF_ACPOVER 自动转换溢出中断

◆返回值: SET/RESET

7.4.11 ADC_Reset函数

◆函数原型: void ADC_Reset(void)

◆功能描述: ADC 复位

◆输入参数: 无

◆返回值: 无

7.5 库函数应用示例

```
void ADCInit(void)  
{  
    ADC_InitStruType x;  
    GPIO_InitSettingType y;  
  
    y.Signal = GPIO_Pin_Signal_Analog;  
    GPIO_Init(GPIO_Pin_A6,&y);  
    x.CLKS = ADC_CLKS_PCLK;                      //ADC 时钟选择
```

```
x.CLKDIV = ADC_CLKDIV_1_1;           //ADC 预分频
x.VREF_SEL = ADC_VREF_SEL_1_8;        //参考电压选择
x.VREFP = ADC_VREFP_IO;                //正参考电压类型选择
x.VREFN = ADC_VREFN_VSS;               //负参考电压类型选择
x.VRBUF_EN = DISABLE;                 //VRBUF 使能

x.CHS = ADC_CHS_AIN1;                  //模拟通道选择
x.SMPS = ADC_SMPS_HARD;                //控制模式选择
x.ST = 8;                              //硬件采样时间
x.VDD5_EN = ENABLE;                   //VDD5 使能
x.CHOP_EN = DISABLE;                  //CHOP EN 使能
x.CHOP_EN1 = DISABLE;                 //CHOP EN1 使能
x.CHOP_CLK_SEL = ADC_CHOP_CLK_SEL_1K; //CHOP 时钟选择
x.CHOP_CLK_DIV = ADC_CHOP_CLK_DIV_1_1; //CHOP 预分频选择
x.FILTERCLK_SEL = ADC_FILTERCLK_SEL_32K; //FILTERCLK 选择

ADC_Init(&x);                          //初始化 ADC
}
```

第8章 液晶显示控制器LCDC

8.1 功能概述

- ◆ 支持 8COM x 28SEG
- ◆ 支持灰度调节功能
- ◆ 支持显示闪烁功能，闪烁频率可调
- ◆ 支持内部偏压电阻可调功能
- ◆ 支持 3 种时钟源选择

8.2 寄存器结构

LCDC 的寄存器定义于文件 HR8P506.h。

```
typedef struct
{
    __IO LCD_CON0_Typedef CON0;
    __IO LCD_TWI_Typedef TWI;
    __IO LCD_SEL_Typedef SEL;
    uint32_t RESERVED0 ;
    __IO LCD_CON1_Typedef CON1;
    uint32_t RESERVED1[3] ;
    __IO LCD_D0_Typedef D0;
    __IO LCD_D1_Typedef D1;
    __IO LCD_D2_Typedef D2;
    __IO LCD_D3_Typedef D3;
    __IO LCD_D4_Typedef D4;
    __IO LCD_D5_Typedef D5;
    __IO LCD_D6_Typedef D6;
} LCD_TypeDef;
```

```
#define APB_BASE (0x40000000UL)
#define LCD_BASE (APB_BASE + 0x01800)
#define LCD ((LCD_TypeDef *) LCD_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。

```
#define LCD_EN()          (LCD->CON1.EN = 1)
#define LCD_DIS()         (LCD->CON1.EN = 0)
#define LCD_RST()         (LCD->CON1.RST = 1)
```

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 LCD_Init函数

- ◆函数原型: `ErrorStatus LCD_Init(LCD_InitStruType * LCD_InitStruct)`
- ◆功能描述: LCD 功能初始化函数
- ◆输入参数: `LCD_InitStruct`: LCD 初始化结构体
- ◆返回值: SUCCESS/ERROR

typedef struct

```
{
    LCD_TYPE_COMS LCD_COMS; /*公共端选择位 */
    uint32_t LCD_SEG; /*段使能 0~0xffffffff */
    LCD_TYPE_VLCD LCD_VLCD; /*外部偏置电压使能位*/
    LCD_TYPE_WFS LCD_WFS; /*驱动波形类型选择*/
    LCD_TYPE_CLK LCD_CLK; /* 时钟选择 */
    uint8_t LCD_PRS; /* 时钟源分频比选择位 0~63*/
    LCD_TYPE_BVS LCD_BVS; /* 灰度 */
    LCD_TYPE_BIAS LCD_BIAS; /* 偏置选择*/
    LCD_TYPE_RT LCD_RT; /* LCD 60K 电阻保持时间选择位 */
    LCD_TYPE_RS LCD_RS; /* LCD 内部偏压电阻总和选择位 */
    LCD_TYPE_FLIK LCD_FLIK; /* LCD 显示闪烁使能 */
    uint8_t LCD_TOFF; /* LCD 熄灭时间=(TOFF+1)*0.25 S 0~255*/
    uint8_t LCD_TON; /* LCD 点亮时间 =(TOFF+1)*0.25 S 0~255*/
}
```

}LCD_InitStruType;

/* 公共端选择位 */

```
typedef enum {
    LCD_COMS_1 = 0, /* ** */
    LCD_COMS_2 = 4, /* COM1~COM0*/
    LCD_COMS_3 = 6, /* COM2~COM0*/
    LCD_COMS_4 = 1, /* COM3~COM0*/
    LCD_COMS_6 = 2, /* COM5~COM0*/
    LCD_COMS_8 = 3, /* COM7~COM0*/
    LCD_COMS_MAX = 6,
}
```

}LCD_TYPE_COMS;

/* 内部外部参考电压选择 */

```
typedef enum {
    LCD_VLCD_INSIDE = 0,
    LCD_VLCD_OUTSIDE = 1,
    LCD_VLCD_MAX = 1,
}
```

}LCD_TYPE_VLCD;

/* LCD 驱动波形类型选择*/

```
typedef enum {
    LCD_WFS_A = 0, /* A 波形(在每一公共端类型内改变相位)*/
}
```

```

        LCD_WFS_B = 1,                                /* B 波形(在每一帧边界改变相位)*/
        LCD_WFS_MAX = 1,
    }LCD_TYPE_WAVE;

```

/* LCD 时钟源选择*/

```

typedef enum {
    LCD_CLK_LRC = 0,                                /* LRC(内部时钟 32KHz)*/
    LCD_CLK_LOSE = 1,                               /* LOSC(外部时钟 32KHz)*/
    LCD_CLK_PCLK1024 = 2,                           /* PCLK1024 分频*/
    LCD_CLK_MAX = 2,
}LCD_TYPE_CLK;

```

/* LCD 显示灰度控制(基于 VDD) */

```

typedef enum {
    LCD_BVS_15_30 = 0,                               /* 1/2VDD*/
    LCD_BVS_16_30 = 1,                               /* 16/30VDD*/
    LCD_BVS_17_30 = 2,                               /* 17/30VDD*/
    LCD_BVS_18_30 = 3,                               /* 18/30VDD*/
    LCD_BVS_19_30 = 4,                               /* 19/30VDD*/
    LCD_BVS_20_30 = 5,                               /* 20/30VDD*/
    LCD_BVS_21_30 = 6,                               /* 21/30VDD*/
    LCD_BVS_22_30 = 7,                               /* 22/30VDD*/
    LCD_BVS_23_30 = 8,                               /* 23/30VDD*/
    LCD_BVS_24_30 = 9,                               /* 24/30VDD*/
    LCD_BVS_25_30 = 10,                              /* 25/30VDD*/
    LCD_BVS_26_30 = 11,                              /* 26/30VDD*/
    LCD_BVS_27_30 = 12,                              /* 27/30VDD*/
    LCD_BVS_28_30 = 13,                              /* 28/30VDD*/
    LCD_BVS_29_30 = 14,                              /* 29/30VDD*/
    LCD_BVS_30_30 = 15,                              /* 30/30VDD*/
    LCD_BVS_MAX = 15,
}LCD_TYPE_BVS;

```

/* LCD 偏置选择*/

```

typedef enum {
    LCD_BIAS_1_2 = 0,                                /* 1/2BIAS*/
    LCD_BIAS_1_3 = 1,                                /* 1/3BIAS*/
    LCD_BIAS_1_4 = 3,                                /* 1/4BIAS*/
    LCD_BIAS_MAX = 3,
}LCD_TYPE_BIAS;

```

/* LCD 内部偏压电阻总和选择*/

```

typedef enum {
    LCD_RS_225K = 0,                                /* 225K 欧姆*/

```

```

    LCD_RS_900K = 1,                /* 900K 欧姆*/
    LCD_RS_60K = 2,                 /* 60K 欧姆*/
    LCD_RS_60K_225K = 4,            /* 60K 与 225K 欧姆自动切换*/
    LCD_RS_60K_900K = 5,            /* 60K 与 900K 欧姆自动切换*/
    LCD_RS_MAX = 5,
}LCD_TYPE_RS;

/* LCD60K 电阻保持时间*/
typedef enum {
    LCD_RT_1_4 = 0,                 /* 1/4 COM 周期 */
    LCD_RT_1_8 = 1,                 /* 1/8 COM 周期 */
    LCD_RT_1_16 = 2,                /* 1/16 COM 周期 */
    LCD_RT_1_32 = 3,                /* 1/32 COM 周期 */
    LCD_RT_1_64 = 4,                /* 1/64 COM 周期 */
    LCD_RT_MAX = 4,
}LCD_TYPE_RT;

/* 闪烁控制 */
typedef enum {
    LCD_FLIK_NO = 0,
    LCD_FLIK_YES = 1,
    LCD_FLIK_MAX = 1,
}LCD_TYPE_FLIK;

```

8.4.2 LCD_GayscaleConfig函数

- ◆函数原型: `ErrorStatus LCD_GayscaleConfig(LCD_TYPE_BVS BVS_Sel)`
- ◆功能描述: LCD 灰度设置
- ◆输入参数: BVS_Sel灰度设置值
- ◆返回值: SUCCESS/ERROR

```

/* LCD 显示灰度控制(基于 VDD) */
typedef enum {
    LCD_BVS_15_30 = 0,              /* 1/2VDD*/
    LCD_BVS_16_30 = 1,              /* 16/30VDD*/
    LCD_BVS_17_30 = 2,              /* 17/30VDD*/
    LCD_BVS_18_30 = 3,              /* 18/30VDD*/
    LCD_BVS_19_30 = 4,              /* 19/30VDD*/
    LCD_BVS_20_30 = 5,              /* 20/30VDD*/
    LCD_BVS_21_30 = 6,              /* 21/30VDD*/
    LCD_BVS_22_30 = 7,              /* 22/30VDD*/
    LCD_BVS_23_30 = 8,              /* 23/30VDD*/
    LCD_BVS_24_30 = 9,              /* 24/30VDD*/
    LCD_BVS_25_30 = 10,             /* 25/30VDD*/
    LCD_BVS_26_30 = 11,             /* 26/30VDD*/
}

```

```
LCD_BVS_27_30 = 12,          /* 27/30VDD*/
LCD_BVS_28_30 = 13,          /* 28/30VDD*/
LCD_BVS_29_30 = 14,          /* 29/30VDD*/
LCD_BVS_30_30 = 15,          /* 30/30VDD*/
LCD_BVS_MAX = 15,
}LCD_TYPE_BVS;
```

8.4.3 LCD_FlickerTimeConfig函数

◆函数原型: `ErrorStatus LCD_FlickerTimeConfig(LCD_TYPE_FLIK Flick,uint8_t On_Time,uint8_t Off_Time)`

◆功能描述: LCD 闪烁功能设置

◆输入参数: Flick 闪烁使能选择
 On_Time 闪烁点亮时间
 Off_Time 闪烁熄灭时间

◆返回值: SUCCESS/ERROR

/* 闪烁控制 */

```
typedef enum {
    LCD_FLIK_NO = 0,
    LCD_FLIK_YES = 1,
    LCD_FLIK_MAX = 1,
}LCD_TYPE_FLIK;
```

On_Time 0~255 (LCDTON+1)*0.25 秒

Off_Time 0~255 (LCDTOFF+1)*0.25 秒

8.4.4 LCD_PixelWriteByte函数

◆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 一次写 8bit 数据

◆输入参数: LCD_DD 像素寄存器选择
 nByte 1/4、2/4、3/4、4/4 8bit 选择
 LCD_data 写入 8bit 数据

◆返回值: SUCCESS/ERROR

/* LCD 像素寄存器选择*/

```
typedef enum
{
    LCD_Pixel_LCDD0 = 0x0 ,          /* 像素寄存器 0 */
    LCD_Pixel_LCDD1 = 0x1 ,          /* 像素寄存器 1 */
    LCD_Pixel_LCDD2 = 0x2 ,          /* 像素寄存器 2 */
    LCD_Pixel_LCDD3 = 0x3 ,          /* 像素寄存器 3 */
    LCD_Pixel_LCDD4 = 0x4 ,          /* 像素寄存器 4 */
}
```



```
LCD_Pixel_LCDD5 = 0x5 ,          /* 像素寄存器 5 */
LCD_Pixel_LCDD6 = 0x6 ,          /* 像素寄存器 6 */
LCD_Pixel_MAX = 0x6,
}LCD_TYPE_PIXEL;
```

/* LCD 单像素寄存器内字节选择 */

```
typedef enum
```

```
{
    LCD_Byte_0 = 0x0 ,            /* 寄存器内第 0 个字节*/
    LCD_Byte_1 = 0x1 ,            /* 寄存器内第 1 个字节*/
    LCD_Byte_2 = 0x2 ,            /* 寄存器内第 2 个字节*/
    LCD_Byte_3 = 0x3 ,            /* 寄存器内第 3 个字节*/
    LCD_Byte_MAX = 0x3,
}LCD_DD_BYTE;
```

8.4.5 LCD_PixelWriteHalfWord函数

◆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 一次写 16bit 数据

◆输入参数: LCD_DD 像素寄存器选择
nHalfWord 前/后 16bit 选择
LCD_data 写入 16bit 数据

◆返回值: SUCCESS/ERROR

/* LCD 单像素寄存器半字选择 */

```
typedef enum
```

```
{
    LCD_HalfWord_0 = 0x0 ,
    LCD_HalfWord_1 = 0x1 ,
    LCD_HalfWord_MAX = 0x1,
}LCD_DD_HALFWORD;
```

8.4.6 LCD_PixelWriteWord函数

◆函数原型: `ErrorStatus LCD_PixelWriteWord(LCD_TYPE_PIXEL LCD_DD, uint32_t LCD_data)`

◆功能描述: LCD 一次写 32bit 数据

◆输入参数: LCD_DD 像素寄存器选择
LCD_data 写入 32bit 数据

◆返回值: SUCCESS/ERROR

8.4.7 LCD_Reset函数

◆函数原型: `void LCD_Reset(void)`

- ◆功能描述：LCD 复位
- ◆输入参数：无
- ◆返回值：无

8.5 库函数应用示例

```
void LCDInit(void)
```

```
{
    GPIO_InitSettingType x;
    LCD_InitStruType y;

    x.Signal = GPIO_Pin_Signal_Analog;
    GPIO_Init(GPIO_Pin_A10,&x);
    GPIO_Init(GPIO_Pin_A11,&x);
    GPIO_Init(GPIO_Pin_A12,&x);
    GPIO_Init(GPIO_Pin_A13,&x);
    GPIO_Init(GPIO_Pin_A14,&x);
    GPIO_Init(GPIO_Pin_A15,&x);
    GPIO_Init(GPIO_Pin_A16,&x);
    GPIO_Init(GPIO_Pin_A17,&x);
    GPIO_Init(GPIO_Pin_A18,&x);
    GPIO_Init(GPIO_Pin_A19,&x);
    GPIO_Init(GPIO_Pin_A20,&x);
    GPIO_Init(GPIO_Pin_A21,&x);

    y.LCD_COMS = LCD_COMS_4;           //公共端选择
    y.LCD_SEG  = 0x000000ff;           //段使能
    y.LCD_VLCD = LCD_VLCD_INSIDE;       //偏置电压选择
    y.LCD_WFS  = LCD_WFS_A;             //驱动波形类型选择
    y.LCD_CLK  = LCD_CLK_LRC;           //时钟选择
    y.LCD_PRS  = 13;                    //时钟源分频比选择
    y.LCD_BVS  = LCD_BVS_30_30;         //灰度选择
    y.LCD_BIAS = LCD_BIAS_1_3;          //偏置选择
    y.LCD_RT   = LCD_RT_1_4;            //内部偏压电阻时间
    y.LCD_RS   = LCD_RS_60K;           //内部偏压电阻选择
    y.LCD_FLIK = LCD_FLIK_NO;          //闪烁使能选择
    y.LCD_TOFF = 0;                     //闪烁熄灭时间
    y.LCD_TON  = 0;                     //闪烁点亮时间

    LCD_Init(&y);                       //初始化 LCD
}
```

第9章 数码管显示控制器(LED C)

9.1 功能概述

- ◆ 支持 1~8 个 8 段式共阴极数码管
- ◆ 支持 3 种时钟源选择

9.2 寄存器结构

LEDC 的寄存器定义于文件 HR8P506.h。

```
typedef struct
{
    __IO LED_CON0_Typedef CON0;
    uint32_t RESERVED0 ;
    __IO LED_SEL_Typedef SEL;
    uint32_t RESERVED1 ;
    __IO LED_CON1_Typedef CON1;
    uint32_t RESERVED2[3] ;
    __IO LED_D0_Typedef D0;
    __IO LED_D1_Typedef D1;
} LED_TypeDef;
#define APB_BASE (0x40000000UL)
#define LED_BASE (APB_BASE + 0x01800)
#define LED ((LED_TypeDef *) LED_BASE )
```

9.3 宏定义

LEDC 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_led.h 中。

```
#define LED_EN()      (LED->CON1.EN = 1)
#define LED_DIS()     (LED->CON1.EN = 0)
#define LED_RST()     (LED->CON1.RST = 1)
```

9.4 库函数

LED 库函数定义于 lib_led.c 中，声明于 lib_led.h 中。

9.4.1 LED_Init函数

- ◆函数原型: `ErrorStatus LED_Init(LED_InitStruType * LED_InitStruct)`
- ◆功能描述: LED 功能初始化函数
- ◆输入参数:
 - LED_InitStruct 初始化结构体
- ◆返回值: SUCCESS 成功
ERROR 失败

```
/* LED 初始化结构体*/
```

```
typedef struct {
    LED_TYPE_MUX LED_MUX;          /* 公共端选择*/
    uint8_t LED_COM;               /* SEG 使能 0~255(每 1bit 控制 1 个 COM) */
    uint8_t LED_SEG;               /* SEG 使能 0~255(每 1bit 控制 1 个 SEG) */
    LED_TYPE_CLK LED_CLK;          /* 时钟选择*/
    uint8_t LED_PRS;               /* 预分频选择 0~63*/
}LED_InitStruType;
```

9.4.2 LED_PixelWriteWord函数

- ◆函数原型: `ErrorStatus LED_PixelWriteWord(LED_TYPE_PIXEL LED_DD,uint32_t LED_data)`
- ◆功能描述: LED 一次写 32bit 数据
- ◆输入参数: LED_DD 像素寄存器选择
LED_data 写入 32bit 数据
- ◆返回值: SUCCESS 成功
ERROR 失败

9.4.3 LED_Reset函数

- ◆函数原型: `void LED_Reset(void)`
- ◆功能描述: LED 复位
- ◆输入参数: 无
- ◆返回值: 无

9.5 库函数应用示例

```
void LEDInit(void)
{
    GPIO_InitSettingType x;
    LED_InitStruType LED_InitStruct;

    LED_InitStruct.LED_CLK = LED_CLK_LRC;          //内部 32K
    LED_InitStruct.LED_COM = 0x0f;                 //com0~com3
    LED_InitStruct.LED_SEG = 0xff;                 //seg0~seg7
    LED_InitStruct.LED_MUX = LED_MUX_0_3;
    LED_InitStruct.LED_PRS = 0;                     //不分频
    LED_Init(&LED_InitStruct);                     //初始化 LED 模块

    x.Dir = GPIO_Direction_Output;
    x.Func = GPIO_Reuse_Func0;
    x.DS = GPIO_DS_Output_Strong;
    x.ODE = GPIO_ODE_Output_Disable ;
    //x.Signal = GPIO_Pin_Signal_Analog;
    x.Signal =GPIO_Pin_Signal_Digital;
```

```
x.PDE = GPIO_PDE_Input_Disable;  
x.PUE = GPIO_PUE_Input_Disable;  
GPIO_Init(GPIO_Pin_A14,&x);  
GPIO_Init(GPIO_Pin_A15,&x);  
GPIO_Init(GPIO_Pin_A16,&x);  
GPIO_Init(GPIO_Pin_A17,&x);  
GPIO_Init(GPIO_Pin_A18,&x);  
GPIO_Init(GPIO_Pin_A19,&x);  
GPIO_Init(GPIO_Pin_A20,&x);  
GPIO_Init(GPIO_Pin_A21,&x);  
GPIO_Init(GPIO_Pin_A13,&x);  
GPIO_Init(GPIO_Pin_A12,&x);  
GPIO_Init(GPIO_Pin_A11,&x);  
GPIO_Init(GPIO_Pin_A10,&x);  
LED_EN();  
}
```

第10章 通用异步收发器UART

10.1 功能概述

- ◆ 支持异步接收和异步发送
- ◆ 支持内置波特率发生器，支持 4 位小数波特率和 12 位整数波特率
- ◆ 兼容 RS-232/RS-442/RS-485 的通讯接口
- ◆ 支持全/半双工通讯模式
- ◆ 异步接收器
- ◆ 异步发送器
- ◆ 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

10.2 寄存器结构

UART 的寄存器定义于文件 HR8P506.h。芯片支持 2 个 UART，为 UART0/1。

typedef struct

```
{  
    __IO UART_CON0_Typedef CON0;  
    __IO UART_CON1_Typedef CON1;  
    uint32_t RESERVED0[2];  
    __IO UART_BRR_Typedef BRR;  
    __I UART_STA_Typedef STA;  
    __IO UART_IE_Typedef IE;  
    __IO UART_IF_Typedef IF;  
    __O UART_TBW_Typedef TBW;  
    __I UART_RBR_Typedef RBR;  
    uint32_t RESERVED1[6];  
    __I UART_TB0_Typedef TB0;  
    __I UART_TB1_Typedef TB1;  
    __I UART_TB2_Typedef TB2;  
    __I UART_TB3_Typedef TB3;  
    __I UART_TB4_Typedef TB4;  
    __I UART_TB5_Typedef TB5;  
    __I UART_TB6_Typedef TB6;  
    __I UART_TB7_Typedef TB7;  
    __I UART_RB0_Typedef RB0;  
    __I UART_RB1_Typedef RB1;  
    __I UART_RB2_Typedef RB2;  
    __I UART_RB3_Typedef RB3;  
    __I UART_RB4_Typedef RB4;  
    __I UART_RB5_Typedef RB5;  
    __I UART_RB6_Typedef RB6;  
    __I UART_RB7_Typedef RB7;  
}
```

```
} UART_TypeDef;
```

```
#define APB_BASE (0x40000000UL)
#define UART0_BASE (APB_BASE + 0x06000)
#define UART1_BASE (APB_BASE + 0x06400)
#define UART0 ((UART_TypeDef *) UART0_BASE)
#define UART1 ((UART_TypeDef *) UART1_BASE)
```

10.3 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。

```
#define UART0_TxEnable() (UART0->CON0.TXEN = 1)
#define UART1_TxEnable() (UART1->CON0.TXEN = 1)
#define UART0_TxDisable() (UART0->CON0.TXEN = 0)
#define UART1_TxDisable() (UART1->CON0.TXEN = 0)
#define UART0_RxEnable() (UART0->CON0.RXEN = 1)
#define UART1_RxEnable() (UART1->CON0.RXEN = 1)
#define UART0_RxDisable() (UART0->CON0.RXEN = 0)
#define UART1_RxDisable() (UART1->CON0.RXEN = 0)

#define UART0_TxRst() (UART0->CON0.TRST = 1)
#define UART1_TxRst() (UART1->CON0.TRST = 1)
#define UART0_RxRst() (UART0->CON0.RRST = 1)
#define UART1_RxRst() (UART1->CON0.RRST = 1)
```

10.4 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。

10.4.1 UART_Init函数

◆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:

IICx: UART0 或 UART1

UART_InitStruct:初始化配置结构体指针

◆返回值: 无

```
typedef struct {
    UART_TYPE_TXFS    UART_StopBits;    /* 停止位选择 */
    UART_TYPE_DATAMOD UART_TxMode;      /* 发送数据格式 */
    UART_TYPE_RTXP    UART_TxPolar;     /* 发送端口极性 */
    UART_TYPE_DATAMOD UART_RxMode;      /* 接收数据格式 */
    UART_TYPE_RTXP    UART_RxPolar;     /* 接收端口极性 */
    uint32_t          UART_BaudRate;    /* 波特率 */
}
```

```

        UART_TYPE_BCS    UART_ClockSet;    /* UART 时钟选择 */
    } UART_InitStruType;

typedef enum {
    UART_StopBits_1 = 0x0,                /* 1 位停止位 */
    UART_StopBits_2 = 0x1,                /* 2 位停止位 */
} UART_TYPE_TXFS;

typedef enum {
    UART_DataMode_7      = 0x4,            /* 7 位数据 */
    UART_DataMode_8      = 0x0,            /* 8 位数据 */
    UART_DataMode_9      = 0x2,            /* 9 位数据 */
    UART_DataMode_7Odd   = 0xD,            /* 7 位数据+奇校验 */
    UART_DataMode_7Even  = 0xC,            /* 7 位数据+偶校验 */
    UART_DataMode_7Add0  = 0xE,            /* 7 位数据+固定 0 */
    UART_DataMode_7Add1  = 0xF,            /* 7 位数据+固定 1 */
    UART_DataMode_8Odd   = 0x9,            /* 8 位数据+奇校验 */
    UART_DataMode_8Even  = 0x8,            /* 8 位数据+偶校验 */
    UART_DataMode_8Add0  = 0xA,            /* 8 位数据+固定 0 */
    UART_DataMode_8Add1  = 0xB,            /* 8 位数据+固定 1 */
} UART_TYPE_DATAMOD;

typedef enum {
    UART_Polar_Normal    = 0x0,            /* 正极性 */
    UART_Polar_Opposite  = 0x1,            /* 负极性 */
} UART_TYPE_RTXP;

typedef enum {
    UART_Clock_1 = 0x1,                    /* PCLK */
    UART_Clock_2 = 0x2,                    /* PCLK / 2 */
    UART_Clock_3 = 0x3,                    /* PCLK / 4 */
    UART_Clock_4 = 0x4,                    /* PCLK / 8 */
} UART_TYPE_BCS;

```

10.4.2 UART_ITConfig函数

- ◆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:
 - IICx: UART0 或 UART1
 - UART_IT: 中断类型
 - NewState: 使能或禁止
- ◆返回值: 无


```
typedef enum {
    UART_IT_TB   = (1 << 0),          /* 发送缓冲器空中断 */
    UART_IT_TC   = (1 << 1),          /* 发送完成中断 */
    UART_IT_TBWE = (1 << 8),          /* 发送缓冲错误中断 */
    UART_IT_TBWO = (1 << 9),          /* 发送缓冲溢出中断 */
    UART_IT_RB   = (1 << 16),         /* 接收缓冲器满中断 */
    UART_IT_ID   = (1 << 17),         /* 接收空闲帧中断 */
    UART_IT_RO   = (1 << 24),         /* 接收数据溢出中断 */
    UART_IT_FE   = (1 << 25),         /* 接收帧错误中断 */
    UART_IT_PE   = (1 << 26),         /* 接收校验错误中断 */
    UART_IT_BDE  = (1 << 27),         /* 波特率检测错误中断 */
    UART_IT_RBRE = (1 << 28),         /* 读接收缓冲错误中断 */
    UART_IT_RBRO = (1 << 29),         /* 读接收缓冲溢出中断 */
} UART_TYPE_IT;
```

10.4.3 UART_TBIMConfig函数

◆函数原型: void UART_TBIMConfig(UART_TypeDef *UARTx, UART_TYPE_TRBIM Type);

◆功能描述: 发送缓冲器空中断类型选择

◆输入参数:

IICx: UART0 或 UART1

Type: 发送缓冲器空中断类型

◆返回值: 无

```
typedef enum {
    UART_TBIM_Byte      = 0x0,        /* 字节空产生中断 */
    UART_TBIM_HalfWord = 0x1,        /* 半字空产生中断 */
    UART_TBIM_Word      = 0x2,        /* 字空产生中断 */
    UART_TBIM_Full      = 0x3,        /* 全空产生中断 */
} UART_TYPE_TRBIM;
```

10.4.4 UART_RBIMConfig函数

◆函数原型: void UART_RBIMConfig(UART_TypeDef *UARTx, UART_TYPE_TRBIM Type);

◆功能描述: 接收缓冲器满中断类型选择

◆输入参数:

IICx: UART0 或 UART1

Type: 接收缓冲器满中断类型, 参见 UART_TYPE_TRBIM 枚举类型

◆返回值: 无

10.4.5 UART_Send函数

◆函数原型:

void UART_SendByte(UART_TypeDef *UARTx, uint8_t data08);

void UART_SendHalfWord(UART_TypeDef *UARTx, uint16_t data16);

void UART_SendWord(UART_TypeDef *UARTx, uint32_t data32);

- ◆功能描述: UART 发送数据
- ◆输入参数:
 - IICx: UART0 或 UART1
 - data08/ data16/ data32: 需要发送的数据
- ◆返回值: 无

10.4.6 UART_Rcv函数

- ◆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 接收数据
- ◆输入参数:
 - IICx: UART0 或 UART1
- ◆返回值: 接收到的数据

10.4.7 UART_GetStatus函数

- ◆函数原型: FlagStatus UART_GetStatus(UART_TypeDef *UARTx, UART_TYPE_STA UART_Flag);
- ◆功能描述: 获取 UART 的状态
- ◆输入参数:
 - IICx: UART0 或 UART1
 - UART_Flag: URAT 的状态标志
- ◆返回值: SET/RESET

```
typedef enum {
    UART_STA_TBOV    = (1 << 4),    /* 发送缓冲器溢出状态位 */
    UART_STA_TXBUSY  = (1 << 5),    /* 发送状态位 */
    UART_STA_RBOV    = (1 << 12),   /* 接收缓冲器溢出状态位*/
    UART_STA_RXBUSY  = (1 << 13),   /* 接收状态位 */
    UART_STA_FER0    = (1 << 16),   /* 当前读取 BYTE0 帧格式错误位 */
    UART_STA_PER0    = (1 << 17),   /* 当前读取 BYTE0 校验错误位 */
    UART_STA_FER1    = (1 << 18),   /* 当前读取 BYTE1 帧格式错误位 */
    UART_STA_PER1    = (1 << 19),   /* 当前读取 BYTE1 校验错误位 */
    UART_STA_FER2    = (1 << 20),   /* 当前读取 BYTE2 帧格式错误位 */
    UART_STA_PER2    = (1 << 21),   /* 当前读取 BYTE2 校验错误位 */
    UART_STA_FER3    = (1 << 22),   /* 当前读取 BYTE3 帧格式错误位 */
    UART_STA_PER3    = (1 << 23),   /* 当前读取 BYTE3 校验错误位 */
} UART_TYPE_STA;
```

10.4.8 UART_GetFlagStatus函数

- ◆函数原型: FlagStatus UART_GetFlagStatus(UART_TypeDef *UARTx, UART_TYPE_FLAG UART_Flag);

◆功能描述：获取中断标志

◆输入参数：

IICx: UART0 或 UART1

UART_Flag: URAT 的中断标志位

◆返回值：SET/RESET

```
typedef enum {
    UART_FLAG_TB    = (1 << 0),      /* 发送缓冲器空中断 */
    UART_FLAG_TC    = (1 << 1),      /* 发送完成中断 */
    UART_FLAG_TBWE  = (1 << 8),      /* 发送缓冲错误中断 */
    UART_FLAG_TBWO  = (1 << 9),      /* 发送缓冲溢出中断 */
    UART_FLAG_RB    = (1 << 16),     /* 接收缓冲器满中断 */
    UART_FLAG_ID    = (1 << 17),     /* 接收空闲帧中断 */
    UART_FLAG_RO    = (1 << 24),     /* 接收数据溢出中断 */
    UART_FLAG_FE    = (1 << 25),     /* 接收帧错误中断 */
    UART_FLAG_PE    = (1 << 26),     /* 接收校验错误中断 */
    UART_FLAG_BDE   = (1 << 27),     /* 波特率检测错误中断 */
    UART_FLAG_RBRE  = (1 << 28),     /* 读接收缓冲错误中断 */
    UART_FLAG_RBRO  = (1 << 29),     /* 读接收缓冲溢出中断 */
} UART_TYPE_FLAG;
```

10.4.9 UART_GetITStatus函数

◆函数原型：ITStatus UART_GetITStatus(UART_TypeDef *UARTx, UART_TYPE_IT UART_Flag);

◆功能描述：获取配置状态

◆输入参数：

IICx: UART0 或 UART1

UART_Flag: URAT 的中断标志位，参见 UART_TYPE_IT 枚举类型

◆返回值：SET/RESET

10.4.10 UART_ClearITPendingBit函数

◆函数原型：void UART_ClearITPendingBit(UART_TypeDef *UARTx, UART_CLR_IF UART_Flag);

◆功能描述：清除中断标志

◆输入参数：

IICx: UART0 或 UART1

UART_Flag: URAT 的中断标志位

◆返回值：SET/RESET

```
typedef enum {
    UART_CLR_TC    = (1 << 1),      /* 发送完成中断 */
    UART_CLR_TBWE  = (1 << 8),      /* 发送缓冲错误中断 */
    UART_CLR_TBWO  = (1 << 9),      /* 发送缓冲溢出中断 */
}
```

```

UART_CLR_ID    = (1 << 17),      /* 接收空闲帧中断 */
UART_CLR_RO    = (1 << 24),      /* 接收数据溢出中断 */
UART_CLR_FE    = (1 << 25),      /* 接收帧错误中断 */
UART_CLR_PE    = (1 << 26),      /* 接收校验错误中断 */
UART_CLR_BDE   = (1 << 27),      /* 波特率检测错误中断 */
UART_CLR_RBRE  = (1 << 28),      /* 读接收缓冲错误中断 */
UART_CLR_RBRO  = (1 << 29),      /* 读接收缓冲溢出中断 */
} UART_CLR_IF;

```

10.5 库函数应用示例

```

void UARTInit(void)
{
    GPIO_InitSettingType x;
    UART_InitStruType y;

    x.Signal = GPIO_Pin_Signal_Digital;
    x.Func = GPIO_Reuse_Func1;
    x.Dir = GPIO_Direction_Output;
    x.ODE = GPIO_ODE_Output_Disable;
    x.DS = GPIO_DS_Output_Strong;
    GPIO_Init(GPIO_Pin_A23, &x);    //TXD - PA23

    x.Signal = GPIO_Pin_Signal_Digital;
    x.Func = GPIO_Reuse_Func1;
    x.Dir = GPIO_Direction_Input;
    x.PUE = GPIO_PUE_Input_Disable;
    x.PDE = GPIO_PDE_Input_Disable;
    GPIO_Init(GPIO_Pin_A22, &x);    //RXD - PA22

    y.UART_BaudRate = 9600;          //设置波特率
    y.UART_ClockSet = UART_Clock_1; //时钟选择: PCLK
    y.UART_RxMode = UART_DataMode_8; //接收数据格式: 8 位数据
    y.UART_RxPolar = UART_Polar_Normal; //接收端口极性: 正常
    y.UART_StopBits = UART_StopBits_1; //停止位: 1
    y.UART_TxMode = UART_DataMode_8; //发送数据格式: 8 位数据
    y.UART_TxPolar = UART_Polar_Normal; //发送端口极性: 正常

    UART_Init(UART0, &y);           //初始化 UART0

    UART_TBIMConfig(UART0, UART_TBIM_Byte); //发送中断模式
    UART_RBIMConfig(UART0, UART_TBIM_Byte); //接收中断模式
    UART_ITConfig(UART0, UART_IT_RB, Enable); //中断配置
    NVIC_Init(NVIC_UART0_IRQn, NVIC_Priority_1, Enable); //中断使能

```

```
    UART0_TxEnable();           //发送使能
    UART0_RxEnable();           //接收使能
}
```

第11章 增强型通用异步收发器EUART

11.1 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

11.2 寄存器结构

EUART 的寄存器定义于文件 HR8P506.h。芯片支持 1 个 EUART，为 EUART0。

typedef struct

```
{
    __IO EUART_CON0_Typedef CON0;
    __IO EUART_CON1_Typedef CON1;
    __IO EUART_CON2_Typedef CON2;
    uint32_t RESERVED0 ;
    __IO EUART_BRR_Typedef BRR;
    uint32_t RESERVED1 ;
    __IO EUART_IE_Typedef IE;
    __IO EUART_IF_Typedef IF;
    __O EUART_TBW_Typedef TBW;
    __I EUART_RBR_Typedef RBR;
    uint32_t RESERVED2[6] ;
    __I EUART_TB01_Typedef TB01;
    __I EUART_TB23_Typedef TB23;
    __I EUART_RB01_Typedef RB01;
    __I EUART_RB23_Typedef RB23;
} EUART_TypeDef;
```

```
#define APB_BASE (0x40000000UL)
```

```
#define EUART0_BASE (APB_BASE + 0x07000)
```

```
#define EUART0 ((EUART_TypeDef *) EUART0_BASE )
```

11.3 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。

```
#define EUART_TxEnable()      (EUART->CON0.TXEN = 1)
#define EUART_TxDisable()    (EUART->CON0.TXEN = 0)
#define EUART_RxEnable()     (EUART->CON1.RXEN = 1)
#define EUART_RxDisable()    (EUART->CON0.RXEN = 0)
```

```
#define EUART_TxRst()          (EUART->CON0.TRST = 1)
#define EUART_RxRst()          (EUART->CON0.RRST = 1)
#define EUART_U7816_REST()     (EUART->CON2.ERST = 1)
```

11.4 库函数

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

11.4.1 EUART_ModeConfig函数

- ◆函数原型：void EUART_ModeConfig(EUART_TYPE_MODE Mode);
- ◆功能描述：配置 EUART 运行模式
- ◆输入参数：
 - Mode: EUART 运行模式
- ◆返回值：无

```
typedef enum {
    EUART_Mode_Uart    = 0x0,      /* 通用 UART 模式 */
    EUART_Mode_U7816   = 0x1,      /* U7816 模式 */
} EUART_TYPE_MODE;
```

11.4.2 EUART_Init函数

- ◆函数原型：void EUART_Init(EUART_InitStruType *EUART_InitStruct);
- ◆功能描述：通用 UART 初始化
- ◆输入参数：
 - EUART_InitStruct: 初始化结构体指针
- ◆返回值：无

```
typedef struct {
    EUART_TYPE_TXFS    EUART_StopBits;    /* 停止位选择 */
    EUART_TYPE_DATAMOD EUART_TxMode;      /* 发送数据格式 */
    EUART_TYPE_RTXP     EUART_TxPolar;     /* 发送端口极性 */
    EUART_TYPE_DATAMOD EUART_RxMode;      /* 接收数据格式 */
    EUART_TYPE_RTXP     EUART_RxPolar;     /* 接收端口极性 */
    uint32_t            EUART_BaudRate;    /* 波特率 */
    EUART_TYPE_BCS      EUART_ClockSet;    /* EUART 时钟选择 */
} EUART_InitStruType;
```

```
typedef enum {
    EUART_DataMode_8      = 0x0,          /* 8 位数据 */
    EUART_DataMode_9      = 0x2,          /* 9 位数据 */
    EUART_DataMode_7      = 0x4,          /* 7 位数据 */
    EUART_DataMode_8Even  = 0x8,          /* 8 位数据+偶校验 */
}
```

```

    EUART_DataMode_8Odd  = 0x9,          /* 8 位数据+奇校验 */
    EUART_DataMode_8Add0 = 0xa,          /* 8 位数据+固定 0 */
    EUART_DataMode_8Add1 = 0xb,          /* 8 位数据+固定 1 */
    EUART_DataMode_7Even = 0xc,          /* 7 位数据+偶校验 */
    EUART_DataMode_7Odd  = 0xd,          /* 7 位数据+奇校验 */
    EUART_DataMode_7Add0 = 0xe,          /* 7 位数据+固定 0 */
    EUART_DataMode_7Add1 = 0xf,          /* 7 位数据+固定 1 */
} EUART_TYPE_DATAMOD;

typedef enum {
    EUART_Polar_Normal    = 0x0,          /* 正极性 */
    EUART_Polar_Opposite = 0x01,         /* 负极性 */
} EUART_TYPE_RTXP;

typedef enum {
    EUART_Clock_1 = 0x1,                  /* PCLK */
    EUART_Clock_2 = 0x2,                  /* PCLK / 2 */
    EUART_Clock_3 = 0x3,                  /* PCLK / 4 */
    EUART_Clock_4 = 0x4,                  /* PCLK / 8 */
} EUART_TYPE_BCS;

```

11.4.3 EUART_BaudConfig函数

- ◆函数原型: void EUART_BaudConfig(uint32_t BaudRate, EUART_TYPE_BCS ClockSet);
- ◆功能描述: 配置 EUART 波特率
- ◆输入参数:
 - BaudRate: 波特率
 - ClockSet: EUART 时钟选择, 参见 EUART_TYPE_BCS 枚举类型
- ◆返回值: 无

11.4.4 EUART_ITConfig函数

- ◆函数原型: void EUART_ITConfig(EUART_TYPE_IT EUART_IT, TYPE_FUNCEN NewState);
- ◆功能描述: 配置 EUART 中断使能
- ◆输入参数:
 - EUART_IT: EUART 中断类型
 - NewState: 使能或禁止
- ◆返回值: 无

```

typedef enum {
    EUART_IT_TB    = (1 << 0),           /* 发送缓冲器空中断 */
    EUART_IT_TC    = (1 << 1),           /* 发送完成中断 */
    EUART_IT_TBWE  = (1 << 8),           /* 写发送缓冲错误中断 */
    EUART_IT_ARTE  = (1 << 12),          /* 自动重发失败中断 */
}

```



```

    EUART_IT_RNA  = (1 << 13),          /* 接收到错误信号中断 */
    EUART_IT_RB   = (1 << 16),          /* 接收缓冲器满中断 */
    EUART_IT_RO   = (1 << 24),          /* 接收缓冲器溢出中断 */
    EUART_IT_FE   = (1 << 25),          /* 接收帧错误中断 */
    EUART_IT_PE   = (1 << 26),          /* 接收校验错误中断 */
    EUART_IT_RBRE = (1 << 28),          /* 读接收缓冲错误中断 */
} EUART_TYPE_IT;

```

11.4.5 EUART_TBIMConfig函数

◆函数原型: void EUART_TBIMConfig(EUART_TYPE_TRBIM Type);

◆功能描述: 配置发送缓冲器空中断类型

◆输入参数:

Type: 发送缓冲器空中断类型

◆返回值: 无

```

typedef enum {
    EUART_TRBIM_Byte      = 0x0,        /* 字节空中断 */
    EUART_TRBIM_HalfWord = 0x1,        /* 半字空中断 */
    EUART_TRBIM_Word      = 0x2,        /* 字空中断 */
} EUART_TYPE_TRBIM;

```

11.4.6 EUART_RBIMConfig函数

◆函数原型: void EUART_RBIMConfig(EUART_TYPE_TRBIM Type);

◆功能描述: 配置接收缓冲器满中断类型

◆输入参数:

Type: 接收缓冲器满中断类型, 参见 EUART_TYPE_TRBIM 枚举类型

◆返回值: 无

11.4.7 EUART_GetFlagStatus函数

◆函数原型: FlagStatus EUART_GetFlagStatus(EUART_TYPE_FLAG EUART_Flag);

◆功能描述: 获取中断标志状态

◆输入参数:

EUART_Flag: 中断标志类型

◆返回值: SET/RESET

```

typedef enum {
    EUART_FLAG_TB   = (1 << 0),          /* 发送缓冲器空中断标志 */
    EUART_FLAG_TC   = (1 << 1),          /* 发送完成中断标志 */
    EUART_FLAG_TBWE = (1 << 8),          /* 写发送缓冲错误中断标志 */
    EUART_FLAG_ARTE = (1 << 12),         /* 自动重发失败中断标志 */
    EUART_FLAG_RNA  = (1 << 13),         /* 接收到错误信号中断标志 */
    EUART_FLAG_RB   = (1 << 16),         /* 接收缓冲器满中断标志 */
} EUART_TYPE_FLAG;

```

```

    EUART_FLAG_RO    = (1 << 24),          /* 接收缓冲器溢出中断标志 */
    EUART_FLAG_FE    = (1 << 25),          /* 接收帧中断标志 */
    EUART_FLAG_PE    = (1 << 26),          /* 接收校验中断标志 */
    EUART_FLAG_RBRE = (1 << 28),          /* 读接收缓冲错误中断标志 */
} EUART_TYPE_FLAG;

```

11.4.8 EUART_GetITStatus函数

- ◆函数原型: ITStatus EUART_GetITStatus(EUART_TYPE_IT EUART_Flag);
- ◆功能描述: 获取中断状态
- ◆输入参数:
 - EUART_Flag: 中断状态类型, 参见 EUART_TYPE_IT 枚举类型
- ◆返回值: SET/RESET

11.4.9 EUART_ClearITPendingBit函数

- ◆函数原型: void EUART_ClearITPendingBit(EUART_CLR_IF EUART_Flag);
- ◆功能描述: 清除中断标志
- ◆输入参数:
 - EUART_Flag: 中断类型
- ◆返回值: 无

```

typedef enum {
    EUART_CLR_TC    = (1 << 1),          /* 发送完成中断标志 */
    EUART_CLR_TBWE = (1 << 8),          /* 写发送缓冲错误中断标志 */
    EUART_CLR_ARTE = (1 << 12),         /* 自动重发失败中断标志 */
    EUART_CLR_RNA  = (1 << 13),         /* 接收到错误信号中断标志 */
    EUART_CLR_RO   = (1 << 24),         /* 接收缓冲器溢出中断标志 */
    EUART_CLR_FE   = (1 << 25),         /* 接收帧中断标志 */
    EUART_CLR_PE   = (1 << 26),         /* 接收校验中断标志 */
    EUART_CLR_RBRE = (1 << 28),         /* 读接收缓冲错误中断标志 */
} EUART_CLR_IF;

```

11.4.10 U7816_Init函数

- ◆函数原型: void U7816_Init(U7816_InitStruType *U7816_InitStruct);
- ◆功能描述: U7816 初始化
- ◆输入参数:
 - U7816_InitStruct: 初始化结构体指针
- ◆返回值: 无

```

typedef struct {
    EUART_TYPE_BCS  U7816_ClockSet;     /* 时钟选择 */
    uint32_t        U7816_BaudRate;     /* 波特率 */
    TYPE_FUNCEN     U7816_ECK0;         /* ECK0 使能 */
}

```

```

    TYPE_FUNCEN      U7816_ECK1;          /* ECK1 使能 */
    U7816_TYPE_CHS    U7816_EIOCh;        /* EIO 通讯通道选择位 */
    U7816_TYPE_IOC    U7816_EIODir;       /* EIO 端口方向 */
    U7816_TYPE_DAS    U7816_DataForm;     /* 数据格式 */
    U7816_TYPE_PS     U7816_DataVerify;   /* 奇偶校验 */
    TYPE_FUNCEN      U7816_AutoRetryTx;   /* 自动重发 */
    TYPE_FUNCEN      U7816_AutoRetryRx;   /* 自动重收 */
    U7816_TYPE_TNAS   U7816_NACK_Width;   /* NACK 信号宽度 */
    U7816_TYPE_ARTS   U7816_RetryTimes;   /* 自动次数 */
    U7816_TYPE_CKS    U7816_CLKS;        /* 时钟源选择 */
    uint8_t          U7816_ETUTime;       /* ETU 保护时间 */
} U7816_InitStruType;

typedef enum {
    U7816_CHS_EIO0 = 0x0,                /* EIO 端口 0 */
    U7816_CHS_EIO1 = 0x1,                /* EIO 端口 1 */
} U7816_TYPE_CHS;

typedef enum {
    U7816_EIODir_In  = 0x0,              /* 接收数据 */
    U7816_EIODir_Out = 0x1,              /* 发送数据 */
} U7816_TYPE_IOC;

typedef enum {
    U7816_DataForm_Normal  = 0x0,        /* 正向 */
    U7816_DataForm_Contrary = 0x1,        /* 反向 */
} U7816_TYPE_DAS;

typedef enum {
    U7816_Verify_Odd  = 0x0,             /* 奇校验 */
    U7816_Verify_Even = 0x1,             /* 偶校验 */
} U7816_TYPE_PS;

typedef enum {
    U7816_NACKWidth_1ETU  = 0x0,         /* 1 个 ETU */
    U7816_NACKWidth_1P5ETU = 0x1,         /* 1.5 个 ETU */
    U7816_NACKWidth_2ETU  = 0x2,         /* 2 个 ETU */
} U7816_TYPE_TNAS;

typedef enum {
    U7816_RetryTime_1 = 0x0,             /* 重发 1 次 */
    U7816_RetryTime_2 = 0x1,             /* 重发 2 次 */
    U7816_RetryTime_3 = 0x2,             /* 重发 3 次 */
    U7816_RetryTime_N = 0x3,             /* 重发无限次, 直到发送成功 */
}

```

```
} U7816_TYPE_ARTS;
```

```
typedef enum {
    U7816_PCLK_1 = 0x0,          /* PCLK   */
    U7816_PCLK_2 = 0x1,          /* PCLK / 2 */
    U7816_PCLK_4 = 0x2,          /* PCLK / 4 */
    U7816_PCLK_8 = 0x3,          /* PCLK / 8 */
} U7816_TYPE_CKS;
```

11. 4. 11 EUART_EIOChConfig函数

- ◆函数原型: void EUART_EIOChConfig (U7816_TYPE_CHS U7816_IO);
- ◆功能描述: EIO 通讯通道选择
- ◆输入参数:
 - U7816_IO: IO 端口号, 参见 U7816_TYPE_CHS 枚举类型
- ◆返回值: 无

11. 4. 12 EUART_EIODirection函数

- ◆函数原型: void EUART_EIODirection (U7816_TYPE_IOC Dir);
- ◆功能描述: EIO 端口方向配置
- ◆输入参数:
 - Dir: 端口方向, 参见 U7816_TYPE_IOC 枚举类型
- ◆返回值: 无

11. 4. 13 EUART_Send函数

- ◆函数原型:


```
void EUART_SendByte (uint8_t ByteData);
void EUART_SendHalfWord (uint16_t HalfWordData);
void EUART_SendWord (uint32_t WordData);
```
- ◆功能描述: 发送数据
- ◆输入参数:
 - ByteData/ HalfWordData/ WordData: 需要发送的数据
- ◆返回值: 无

11. 4. 14 U7816_Rcv函数

- ◆函数原型:


```
uint8_t EUART_RecByte (void);
uint16_t EUART_RecHalfWord (void);
uint32_t EUART_RecWord (void);
```
- ◆功能描述: 接收数据
- ◆输入参数: 无
- ◆返回值: 接收到的数据

11.5 库函数应用示例

```
void User_EuartConfig(void)
```

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

第12章 IIC串行总线

12.1 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 SCL0 和 SDA0，均支持输出模式可配置：推挽输出或开漏输出
- ◆ 通讯端口 SCL0 和 SDA0 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

12.2 寄存器结构

IIC 的寄存器定义于文件 HR8P506.h。芯片支持 1 个 IIC。

```
typedef struct
{
    __IO I2C_CON_Typedef CON;
    __IO I2C_MOD_Typedef MOD;
    __IO I2C_IE_Typedef IE;
    __IO I2C_IF_Typedef IF;
    __O I2C_TBW_Typedef TBW;
    __I I2C_RBR_Typedef RBR;
    __I I2C_TB_Typedef TB;
    __I I2C_RB_Typedef RB;
    __I I2C_STA_Typedef STA;
} I2C_TypeDef;

#define APB_BASE (0x40000000UL)
#define I2C0_BASE (APB_BASE + 0x09000)
#define I2C0 ((I2C_TypeDef *) I2C0_BASE )
```

12.3 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。

```
#define IIC0_Enable()      (I2C0->CON.EN = 1)
#define IIC1_Enable()      (I2C1->CON.EN = 1)
#define IIC0_Disable()     (I2C0->CON.EN = 0)
#define IIC1_Disable()     (I2C1->CON.EN = 0)

#define IIC0_Reset()       (I2C0->CON.RST = 1)
#define IIC1_Reset()       (I2C1->CON.RST = 1)

#define IIC0_TJEnable()    (I2C0->CON.TJE = 1)
#define IIC1_TJEnable()    (I2C1->CON.TJE = 1)
#define IIC0_TJDisable()   (I2C0->CON.TJE = 0)
#define IIC1_TJDisable()   (I2C1->CON.TJE = 0)

#define IIC0_Read()        (I2C0->CON.RW = 1)
#define IIC1_Read()        (I2C1->CON.RW = 1)
#define IIC0_Write()       (I2C0->CON.RW = 0)
#define IIC1_Write()       (I2C1->CON.RW = 0)

#define IIC0_CSEnable()    (I2C0->MOD.CSE = 1)
#define IIC1_CSEnable()    (I2C1->MOD.CSE = 1)
#define IIC0_CSDisable()   (I2C0->MOD.CSE = 0)
#define IIC1_CSDisable()   (I2C1->MOD.CSE = 0)

#define IIC0_ANAEnable()   (I2C0->MOD.ANAE = 1)
#define IIC1_ANAEnable()   (I2C1->MOD.ANAE = 1)
#define IIC0_ANADisable()  (I2C0->MOD.ANAE = 0)
#define IIC1_ANADisable()  (I2C1->MOD.ANAE = 0)

#define IIC0_SRAEnable()   (I2C0->MOD.SRAE = 1)
#define IIC1_SRAEnable()   (I2C1->MOD.SRAE = 1)
#define IIC0_SRADisable()  (I2C0->MOD.SRAE = 0)
#define IIC1_SRADisable()  (I2C1->MOD.SRAE = 0)

#define IIC0_SPANable()    (I2C0->MOD.SPAP = 1)
#define IIC1_SPANable()    (I2C1->MOD.SPAP = 1)
#define IIC0_SPADisable()  (I2C0->MOD.SPAP = 0)
#define IIC1_SPADisable()  (I2C1->MOD.SPAP = 0)

#define IIC0_SRTrigger()   (I2C0->MOD.SRT = 1)
#define IIC1_SRTrigger()   (I2C1->MOD.SRT = 1)
```

```
#define IIC0_SPTTrigger() (I2C0->MOD.SPT = 1)
#define IIC1_SPTTrigger() (I2C1->MOD.SPT = 1)

#define IIC0_RDTrigger() (I2C0->MOD.RDT = 1)
#define IIC1_RDTrigger() (I2C1->MOD.RDT = 1)

#define IIC0_Release()      (I2C0->MOD.BLD = 1)
#define IIC1_Release()      (I2C1->MOD.BLD = 1)

#define IIC0_TACK()         (I2C0->MOD.TAS = 1)
#define IIC1_TACK()         (I2C1->MOD.TAS = 1)
#define IIC0_TNACK()        (I2C0->MOD.TAS = 0)
#define IIC1_TNACK()        (I2C1->MOD.TAS = 0)
```

12.4 库函数

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

12.4.1 IIC_Init函数

- ◆函数原型：void IIC_Init(I2C_TypeDef *IICx, IIC_InitStruType *IIC_InitStruct);
- ◆功能描述：IIC 初始化
- ◆输入参数：
 - IICx: I2C0
 - IIC_InitStruct:初始化配置结构体指针
- ◆返回值：无

初始化配置结构体：

```
typedef struct {
    IIC_TYPE_PINOD    IIC_SckOd;           /* SCK 输出模式 */
    IIC_TYPE_PINOD    IIC_SdaOd;           /* SDA 输出模式 */
    TYPE_FUNCEN       IIC_16XSamp;         /* 16 倍采样使能位 */
    uint32_t           IIC_Clk;             /* IIC 频率 */
    IIC_TYPE_MODE      IIC_Mode;            /* IIC 工作模式 */
    TYPE_FUNCEN        IIC_AutoStop;        /* 自动停止 */
    TYPE_FUNCEN        IIC_AutoCall;        /* 自动寻呼 */
} IIC_InitStruType;
```

引脚开漏设置枚举类型：

```
typedef enum {
    IIC_PinMode_PP = 0x0,                  /* 推挽 */
    IIC_PinMode_OD = 0x1,                  /* 开漏 */
} IIC_TYPE_PINOD;
```

工作模式选择枚举类型：

```
typedef enum {
```



```
IIC_Mode_Master = 0x0,          /* 主动 */
IIC_Mode_Slave   = 0x1,          /* 从动 */
} IIC_TYPE_MODE;
```

12.4.2 IIC_ITConfig函数

◆函数原型: void IIC_ITConfig(I2C_TypeDef *IICx, IIC_TYPE_IT IIC_IT, TYPE_FUNCEN NewState);

◆功能描述: 配置 IIC 中断

◆输入参数:

IICx: I2C0

IIC_IT: 中断类型

NewState: 使能或禁止

◆返回值: 无

中断选择枚举类型:

```
typedef enum {
    IIC_IT_SR    = (1 << 0),      /* 起始位中断 */
    IIC_IT_SP    = (1 << 1),      /* 停止位中断 */
    IIC_IT_TB    = (1 << 2),      /* 发送缓冲空中断 */
    IIC_IT_RB    = (1 << 3),      /* 接收缓冲满中断 */
    IIC_IT_TE    = (1 << 4),      /* 发送数据错误中断 */
    IIC_IT_RO    = (1 << 5),      /* 接收数据溢出中断 */
    IIC_IT_NA    = (1 << 6),      /* 未应答 NACK 中断 */
    IIC_IT_TBWE  = (1 << 7),      /* 发送数据写错误中断 */
    IIC_IT_TIDLE = (1 << 12),     /* 空闲中断中断 */
} IIC_TYPE_IT;
```

12.4.3 IIC_SendAddress函数

◆函数原型: void IIC_SendAddress(I2C_TypeDef *IICx, uint8_t IIC_Address, IIC_TYPE_RWMODE Mode);

◆功能描述: 发送从机地址（主控模式）

◆输入参数:

IICx: I2C0

IIC_Address: 从机地址

Mode: 读或写

◆返回值: 无

读写模式枚举类型:

```
typedef enum {
    IIC_Mode_Write = 0x0,          /* 写 */
    IIC_Mode_Read  = 0x1,          /* 读 */
} IIC_TYPE_RWMODE;
```

12.4.4 IIC_SetAddress函数

- ◆函数原型: void IIC_SetAddress(I2C_TypeDef *IICx, uint8_t IIC_Address);
- ◆功能描述: 设置地址 (从动模式)
- ◆输入参数:
 - IICx: I2C0
 - IIC_Address: 从机地址
- ◆返回值: 无

12.4.5 IIC_RecModeConfig函数

- ◆函数原型: void IIC_RecModeConfig(I2C_TypeDef *IICx, IIC_TYPE_RECMode RecType);
- ◆功能描述: 设置地址 (从动模式)
- ◆输入参数:
 - IICx: I2C0
 - RecType: 接收模式
- ◆返回值: 无

```
typedef enum {  
    IIC_RecMode_0 = 0x0,    /* 接收 1 个字节, 发送 ACK */  
    IIC_RecMode_1 = 0x1,    /* 接收 1 个字节, 发送 NACK */  
    IIC_RecMode_2 = 0x2,    /* 接收 2 个字节, 发送 ACK */  
    IIC_RecMode_3 = 0x3,    /* 接收 2 个字节, 前发送 ACK, 后发送 NACK */  
    IIC_RecMode_4 = 0x4,    /* 接收 4 个字节, 每字节发送 ACK */  
    IIC_RecMode_5 = 0x5,    /* 接收 4 个字节, 前 3 字节发送 ACK, 后 NACK */  
} IIC_TYPE_RECMode;
```

12.4.6 IIC_TBIMConfig函数

- ◆函数原型: void IIC_TBIMConfig(I2C_TypeDef *IICx, IIC_TYPE_TRBIM Type);
- ◆功能描述: 发送缓冲器空中断模式
- ◆输入参数:
 - IICx: I2C0
 - Type: 中断模式
- ◆返回值: 无

```
typedef enum {  
    IIC_TRBIM_Byte      = 0x0,    /* 字节空中断*/  
    IIC_TRBIM_HalfWord = 0x1,    /* 半字空中断 */  
    IIC_TRBIM_Word      = 0x2,    /* 字空中断 */  
} IIC_TYPE_TRBIM;
```

12.4.7 IIC_RBIMConfig函数

- ◆函数原型: void IIC_RBIMConfig(I2C_TypeDef *IICx, IIC_TYPE_TRBIM Type);
- ◆功能描述: 接收缓冲器空中断模式

◆输入参数:

IICx: I2C0

Type: 中断模式, 参考 IIC_TYPE_TRBIM 枚举类型

◆返回值: 无

12.4.8 IIC_AckDelay函数

◆函数原型: void IIC_AckDelay(I2C_TypeDef *IICx, IIC_TYPE_ADLY Type, TYPE_FUNCEN NewStatus);

◆功能描述: 应答延迟配置

◆输入参数:

IICx: I2C0

Type: 延迟时间

NewState: 使能或禁止

◆返回值: 无

延迟时间枚举类型:

```
typedef enum {
    IIC_AckDelay_0P5 = 0x0,    /* 0.5 个时钟周期*/
    IIC_AckDelay_1    = 0x1,    /* 1 个时钟周期*/
    IIC_AckDelay_1P5 = 0x2,    /* 1.5 个时钟周期*/
    IIC_AckDelay_2    = 0x3,    /* 2 个时钟周期*/
    IIC_AckDelay_2P5 = 0x4,    /* 2.5 个时钟周期*/
    IIC_AckDelay_3    = 0x5,    /* 3 个时钟周期*/
    IIC_AckDelay_3P5 = 0x6,    /* 3.5 个时钟周期*/
    IIC_AckDelay_4    = 0x7,    /* 4 个时钟周期*/
} IIC_TYPE_ADLY;
```

12.4.9 IIC_TISConfig函数

◆函数原型: void IIC_TISConfig(I2C_TypeDef *IICx, IIC_TYPE_TIS Time);

◆功能描述: 数据帧传输间隔设置

◆输入参数:

IICx: I2C0

Time: 传输间隔

◆返回值: 无

```
typedef enum {
    IIC_TIS_Disable = 0x0,    /* 传输间隔 0*/
    IIC_TIS_1       = 0x1,    /* 传输间隔 1*/
    IIC_TIS_2       = 0x2,    /* 传输间隔 2*/
    IIC_TIS_3       = 0x3,    /* 传输间隔 3*/
    IIC_TIS_4       = 0x4,    /* 传输间隔 4*/
    IIC_TIS_5       = 0x5,    /* 传输间隔 5*/
    IIC_TIS_6       = 0x6,    /* 传输间隔 6*/
}
```

```

IIC_TIS_7      = 0x7,      /* 传输间隔 7*/
IIC_TIS_8      = 0x8,      /* 传输间隔 8*/
IIC_TIS_9      = 0x9,      /* 传输间隔 9*/
IIC_TIS_A      = 0xA,      /* 传输间隔 10*/
IIC_TIS_B      = 0xB,      /* 传输间隔 11*/
IIC_TIS_C      = 0xC,      /* 传输间隔 12*/
IIC_TIS_D      = 0xD,      /* 传输间隔 13*/
IIC_TIS_E      = 0xE,      /* 传输间隔 14*/
IIC_TIS_F      = 0xF,      /* 传输间隔 15*/
} IIC_TYPE_TIS;

```

12. 4. 10 IIC_Send函数

◆函数原型:

```

void IIC_SendByte(I2C_TypeDef *IICx, uint8_t Byte);
void IIC_SendHalfWord(I2C_TypeDef *IICx, uint16_t HalfWord);
void IIC_SendWord(I2C_TypeDef *IICx, uint32_t Word);

```

◆功能描述: 发送字节、半字、字

◆输入参数:

IICx: I2C0

Byte/ HalfWord/ Word: 要发送的数据

◆返回值: 无

12. 4. 11 IIC_Rcv函数

◆函数原型:

```

uint8_t IIC_RecByte(I2C_TypeDef *IICx);
uint16_t IIC_RecHalfWord(I2C_TypeDef *IICx);
uint32_t IIC_RecWord(I2C_TypeDef *IICx);

```

◆功能描述: 接收字节、半字、字

◆输入参数:

IICx: I2C0

◆返回值: 接收到的数据

12. 4. 12 IIC_GetRWMode函数

◆函数原型: IIC_TYPE_RWMODE IIC_GetRWMode(I2C_TypeDef *IICx);

◆功能描述: 获取 IIC 读写状态

◆输入参数:

IICx: I2C0

◆返回值: 读或写, 参见 IIC_TYPE_RWMODE 枚举类型

12. 4. 13 IIC_GetTBStatus函数

◆函数原型: FlagStatus IIC_GetTBStatus(I2C_TypeDef *IICx);

◆功能描述: 获取发送缓冲器状态, TB0-TB3 全空返回 SET, 否则返回 RESET

◆输入参数:

IICx: I2C0

◆返回值: SET/RESET

12. 4. 14 IIC_GetFlagStatus函数

◆函数原型: FlagStatus IIC_GetFlagStatus(I2C_TypeDef *IICx, IIC_TYPE_IF IIC_Flag);

◆功能描述: 获取中断标志位

◆输入参数:

IICx: I2C0

IIC_Flag: 中断类型,

◆返回值: SET/RESET

```
typedef enum {
    IIC_IF_SR      = (1 << 0),    /* 起始位中断*/
    IIC_IF_SP      = (1 << 1),    /* 停止位中断*/
    IIC_IF_TB      = (1 << 2),    /* 发送缓冲器空位中断*/
    IIC_IF_RB      = (1 << 3),    /* 接收缓冲器满中断*/
    IIC_IF_TE      = (1 << 4),    /* 发送数据错误中断*/
    IIC_IF_RO      = (1 << 5),    /* 接收数据溢出中断*/
    IIC_IF_NA      = (1 << 6),    /* 未应答 NACK 中断*/
    IIC_IF_TBWE    = (1 << 7),    /* 发送数据写错误中断*/
    IIC_IF_TIDLE   = (1 << 12),   /* 空闲中断*/
} IIC_TYPE_IF;
```

12. 4. 15 IIC_GetITStatus函数

◆函数原型: FlagStatus IIC_GetITStatus(I2C_TypeDef *IICx, IIC_TYPE_IT IIC_Flag);

◆功能描述: 获取中断配置状态

◆输入参数:

IICx: I2C0

IIC_Flag: 中断类型

◆返回值: SET/RESET

```
typedef enum {
    IIC_IT_SR      = (1 << 0),    /* 起始位中断*/
    IIC_IT_SP      = (1 << 1),    /* 停止位中断*/
    IIC_IT_TB      = (1 << 2),    /* 发送缓冲器空位中断*/
    IIC_IT_RB      = (1 << 3),    /* 接收缓冲器满中断*/
    IIC_IT_TE      = (1 << 4),    /* 发送数据错误中断*/
    IIC_IT_RO      = (1 << 5),    /* 接收数据溢出中断*/
    IIC_IT_NA      = (1 << 6),    /* 未应答 NACK 中断*/
    IIC_IT_TBWE    = (1 << 7),    /* 发送数据写错误中断*/
    IIC_IT_TIDLE   = (1 << 12),   /* 空闲中断*/
} IIC_TYPE_IT;
```

12. 4. 16 IIC_ClearITPendingBit函数

◆函数原型: void IIC_ClearITPendingBit(I2C_TypeDef *IICx, IIC_CLR_IF IIC_IT);;

◆功能描述: 清中断标志

◆输入参数:

IICx: I2C0

IIC_IT: 中断类型

◆返回值: 无

```
typedef enum {
    IIC_CLR_SR      = (1 << 0),      /* 起始位中断*/
    IIC_CLR_SP      = (1 << 1),      /* 停止位中断*/
    IIC_CLR_TE      = (1 << 4),      /* 发送数据错误中断*/
    IIC_CLR_RO      = (1 << 5),      /* 接收数据溢出中断*/
    IIC_CLR_NA      = (1 << 6),      /* 未应答 NACK 中断*/
    IIC_CLR_TBWE     = (1 << 7),      /* 发送数据写错误中断*/
    IIC_CLR_TIDLE    = (1 << 12),     /* 空闲中断*/
} IIC_CLR_IF;
```

12. 5 库函数应用示例

```
void IIC0_MasterInit(void)
{
    GPIO_InitSettingType x;
    IIC_InitStruType y;                //定义结构体

    //IIC0  PB02 03
    x.Func = GPIO_Reuse_Func3;
    x.ODE   = GPIO_ODE_Output_Disable;
    x.DS     = GPIO_DS_Output_Normal;
    GPIO_Init(GPIO_Pin_B2, &x);        //SCL  - PA24
    GPIO_Init(GPIO_Pin_B3, &x);        //SDA  - PA25
    y.IIC_16XSamp = Disable;           //16 倍采样设置
    y.IIC_AutoStop = Disable;          //自动停止
    y.IIC_Clk      = 300000;           //通信频率 300KHz
    y.IIC_Mode     = IIC_Mode_Master;  //主从模式
    y.IIC_SckOd    = IIC_PinMode_OD;   //端口开漏
    y.IIC_SdaOd    = IIC_PinMode_OD;   //端口开漏
    IIC_Init(I2C0, &y);                //初始化 IIC
    IIC_TBIMConfig(I2C0, IIC_TRBIM_Byte); //发送中断模式
    IIC_RBIMConfig(I2C0, IIC_TRBIM_Byte); //接收中断模式
    IIC_RecModeConfig(I2C0, IIC_RecMode_0); //接收模式配置
    IIC0_Enable();                     //使能 IIC0
}
```

第13章 SPI串行总线

13.1 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 1 到 8 位帧位宽选择
- ◆ 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

13.2 寄存器结构

SPI 的寄存器定义于文件 HR8P506.h。芯片支持 2 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;
    __IO SPI_CKS_Typedef CKS;
} SPI_TypeDef;
```

```
#define APB_BASE (0x40000000UL)
#define SPI0_BASE (APB_BASE + 0x08000)
#define SPI1_BASE (APB_BASE + 0x08400)
#define SPI0 ((SPI_TypeDef *) SPI0_BASE )
#define SPI1 ((SPI_TypeDef *) SPI1_BASE )
```

13.3 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。

```
/* SPI0 */
#define SPI0_Enable()      (SPI0->CON.EN  = 1)
#define SPI0_Disable()     (SPI0->CON.EN  = 0)
#define SPI0_RecEnable()   (SPI0->CON.REN = 1)
#define SPI0_RecDisable()  (SPI0->CON.REN = 0)
```

```
#define SPI0_Rst()          (SPI0->CON.RST = 1)
```

```
/* SPI1 */
```

```
#define SPI1_Enable()      (SPI1->CON.EN = 1)
```

```
#define SPI1_Disable()     (SPI1->CON.EN = 0)
```

```
#define SPI1_RecEnable() (SPI1->CON.REN = 1)
```

```
#define SPI1_RecDisable() (SPI1->CON.REN = 0)
```

```
#define SPI1_Rst()         (SPI1->CON.RST = 1)
```

13.4 库函数

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

13.4.1 SPI_Init函数

◆函数原型:

```
void SPI_Init(SPI_TypeDef *SPIdx, SPI_InitStruType *SPI_InitStruct);
```

◆功能描述: SPI 初始化

◆输入参数:

IICx: SPI 或 SPI1

SPI_InitStruct: SPI 初始化结构体指针

◆返回值: 无

```
typedef struct {
    uint32_t SPI_Freq;          /* SPI 频率*/
    SPI_TYPE_DFS SPI_Df;       /* 数据格式*/
    SPI_TYPE_MODE SPI_Mode;     /* 通讯模式*/
    uint8_t SPI_DW;             /* 发送帧位宽*/
    TYPE_FUNCEN SPI_DelayRec;   /* 延迟接收使能*/
    TYPE_FUNCEN SPI_DelaySend;  /* 发送间隔使能*/
    uint8_t SPI_SendDelayPeroid; /* 发送间隔周期*/
} SPI_InitStruType;

typedef enum {
    SPI_RiseSendFallRec = 0x0, /* 上升沿发送, 下降沿接收*/
    SPI_FallSendRiseRec = 0x1, /* 下降沿发送, 上升沿接收*/
    SPI_RiseRecFallSend = 0x2, /* 上升沿接收, 下降沿发送*/
    SPI_FallRecRiseSend = 0x3, /* 下降沿接收, 上升沿发送*/
} SPI_TYPE_DFS;

typedef enum {
    SPI_Mode_Master = 0x0, /* 主动模式*/
    SPI_Mode_Slave = 0x1, /* 从动模式*/
} SPI_TYPE_MODE;
```


13.4.2 SPI_ITConfig函数

◆函数原型: void SPI_ITConfig(SPI_TypeDef *SPIx, SPI_TYPE_IT SPI_IE, TYPE_FUNCEN NewState);

◆功能描述: SPI 中断配置

◆输入参数:

IICx: SPI 或 SPI1

SPI_IE: SPI 中断类型

NewState: 使能或禁止

◆返回值: 无

```
typedef enum {
    SPI_IT_TB   = (1 << 0),      /* 发送缓冲器空中断*/
    SPI_IT_RB   = (1 << 1),      /* 接收缓冲器满中断*/
    SPI_IT_TE   = (1 << 2),      /* 发送数据错误中断*/
    SPI_IT_RO   = (1 << 3),      /* 接收错误溢出中断*/
    SPI_IT_ID   = (1 << 4),      /* 空闲状态中断*/
    SPI_IT_NSS  = (1 << 5),      /* 片选变化中断*/
    SPI_IT_TBWE = (1 << 6),      /* 发送数据写错误中断*/
} SPI_TYPE_IT;
```

13.4.3 SPI_DataFormatConfig函数

◆函数原型: void SPI_DataFormatConfig(SPI_TypeDef *SPIx, SPI_TYPE_DFS Type);

◆功能描述: SPI 数据格式配置

◆输入参数:

IICx: SPI 或 SPI1

Type: 数据格式类型类型, 参考 SPI_TYPE_DFS 枚举类型

◆返回值: 无

13.4.4 SPI_Send函数

◆函数原型:

void SPI_SendByte(SPI_TypeDef *SPIx, uint8_t Temp);

void SPI_SendHalfWord(SPI_TypeDef *SPIx, uint16_t Temp);

void SPI_SendWord(SPI_TypeDef *SPIx, uint32_t Temp);

◆功能描述: SPI 发送数据

◆输入参数:

IICx: SPI 或 SPI1

Temp: 需要发送的数据

◆返回值: 无

13.4.5 SPI_Rcv函数

◆函数原型:

uint8_t SPI_RecByte(SPI_TypeDef *SPIx);

uint16_t SPI_RecHalfWord(SPI_TypeDef *SPIx);

uint32_t SPI_RecWord(SPI_TypeDef *SPIx);

◆功能描述: SPI 接收数据

◆输入参数:

IICx: SPI 或 SPI1

◆返回值: 接收到的数据

13.4.6 SPI_TBIMConfig函数

◆函数原型: void SPI_TBIMConfig(SPI_TypeDef *SPIx, SPI_TYPE_TRBIM Type);

◆功能描述: 发送缓冲器空中断模式选择

◆输入参数:

IICx: SPI 或 SPI1

Type: 中断类型

◆返回值: 无

```
typedef enum {
    SPI_IType_BYTE      = 0x0,      /* 字节空中断*/
    SPI_IType_HALFWORD = 0x1,      /* 半字空中断*/
    SPI_IType_WORD      = 0x2,      /* 字空中断*/
} SPI_TYPE_TRBIM;
```

13.4.7 SPI_RBIMConfig函数

◆函数原型: void SPI_RBIMConfig(SPI_TypeDef *SPIx, SPI_TYPE_TRBIM Type);

◆功能描述: 接收缓冲器满中断模式选择

◆输入参数:

IICx: SPI 或 SPI1

Type: 中断类型, 参见 SPI_TYPE_TRBIM 枚举类型

◆返回值: 无

13.4.8 SPI_GetFlagStatus函数

◆函数原型: FlagStatus SPI_GetFlagStatus(SPI_TypeDef *SPIx, SPI_TYPE_FLAG Flag);

◆功能描述: 获取 SPI 中断标志状态

◆输入参数:

IICx: SPI 或 SPI1

Flag: 中断类型

◆返回值: SET/RESET

```
typedef enum {
    SPI_Flag_TB      = (1 << 0),      /* 发送缓冲器空中断标志*/
    SPI_Flag_RB      = (1 << 1),      /* 接收缓冲器满中断标志*/
    SPI_Flag_TE      = (1 << 2),      /* 发送错误中断标志, 仅从动模式*/
    SPI_Flag_RO      = (1 << 3),      /* 接收错误溢出中断标志*/
}
```

```

SPI_Flag_ID      = (1 << 4),      /* 空闲状态中断标志*/
SPI_Flag_NSSIF = (1 << 5),      /* 片选变化中断标志*/
SPI_Flag_TBWE   = (1 << 6),      /* 发送数据写错误中断标志*/
SPI_Flag_NSS    = (1 << 7),      /* 片选标志位*/
SPI_Flag_TBEO0  = (1 << 8),      /* TB0 空标志位*/
SPI_Flag_TBEO1  = (1 << 9),      /* TB1 空标志位*/
SPI_Flag_TBEO2  = (1 << 10),     /* TB2 空标志位*/
SPI_Flag_TBEO3  = (1 << 11),     /* TB3 空标志位*/
SPI_Flag_RBFF0  = (1 << 12),     /* RB0 满标志位*/
SPI_Flag_RBFF1  = (1 << 13),     /* RB1 满标志位*/
SPI_Flag_RBFF2  = (1 << 14),     /* RB2 满标志位*/
SPI_Flag_RBFF3  = (1 << 15),     /* RB3 满标志位*/
SPI_Flag_IDLE   = (1 << 16),     /* 空闲标志位*/
SPI_Flag_TMS    = (1 << 17),     /* 帧发送间隔状态标志位*/
}SPI_TYPE_FLAG;

```

13. 4. 9 SPI_GetITStatus函数

- ◆函数原型: ITStatus SPI_GetITStatus(SPI_TypeDef *SPIx, SPI_TYPE_IT Flag);
- ◆功能描述: 获取 SPI 中断状态
- ◆输入参数:
 - IICx: SPI 或 SPI1
 - Flag: 中断类型, 参见 SPI_TYPE_IT 枚举类型
- ◆返回值: SET/RESET

13. 4. 10 SPI_GetStatus函数

- ◆函数原型: FlagStatus SPI_GetStatus(SPI_TypeDef *SPIx, SPI_TYPE_STA Flag);
- ◆功能描述: 获取 SPI 状态
- ◆输入参数:
 - IICx: SPI 或 SPI1
 - Flag: SPI 状态类型
- ◆返回值: SET/RESET

```

typedef enum {
    SPI_STA_NSS      = (1 << 7),      /* 片选标志位*/
    SPI_STA_TBEO0    = (1 << 8),      /* TB0 空标志位*/
    SPI_STA_TBEO1    = (1 << 9),      /* TB1 空标志位*/
    SPI_STA_TBEO2    = (1 << 10),     /* TB2 空标志位*/
    SPI_STA_TBEO3    = (1 << 11),     /* TB3 空标志位*/
    SPI_STA_RBFF0    = (1 << 12),     /* RB0 满标志位*/
    SPI_STA_RBFF1    = (1 << 13),     /* RB1 满标志位*/
    SPI_STA_RBFF2    = (1 << 14),     /* RB2 满标志位*/
    SPI_STA_RBFF3    = (1 << 15),     /* RB3 满标志位*/
    SPI_STA_IDLE     = (1 << 16),     /*空闲状态标志位*/
}

```

```
} SPI_TYPE_STA;
```

13.4.11 SPI_ClearITPendingBit函数

◆函数原型: void SPI_ClearITPendingBit(SPI_TypeDef *SPIx, SPI_CLR_IF Flag);

◆功能描述: 清除中断标志位

◆输入参数:

IICx: SPI 或 SPI1

Flag: SPI 状态类型

◆返回值: 无

```
typedef enum {
    SPI_CLR_TE    = (1 << 2),    /* 发送错误中断标志, 仅从动模式*/
    SPI_CLR_RO    = (1 << 3),    /* 接收错误溢出中断标志*/
    SPI_CLR_ID    = (1 << 4),    /* 空闲状态中断标志*/
    SPI_CLR_NSS   = (1 << 5),    /* 片选变化中断标志*/
    SPI_CLR_TBWE  = (1 << 6),    /* 发送数据写错误中断标志*/
} SPI_CLR_IF;
```

13.4.12 Clear_TBW函数

◆函数原型: void Clear_TBW(SPI_TypeDef *SPIx);

◆功能描述: 清空发送缓冲器

◆输入参数:

IICx: SPI 或 SPI1

◆返回值: 无

13.4.13 Clear_RBR函数

◆函数原型: void Clear_RBR(SPI_TypeDef *SPIx);

◆功能描述: 清空接收缓冲器

◆输入参数:

IICx: SPI 或 SPI1

◆返回值: 无

13.5 库函数应用示例

```
void SPI1Init(void)
{
    GPIO_InitSettingType x;
    SPI_InitStruType y;                //定义结构体

    /* PA 24 25 26 27 */
    x.Func = GPIO_Reuse_Func1;
    x.Dir  = GPIO_Direction_Output;
```

```
x.ODE = GPIO_ODE_Output_Disable;
x.DS = GPIO_DS_Output_Strong;
GPIO_Init(GPIO_Pin_A24, &x);           //DO - PA24
GPIO_Init(GPIO_Pin_A25, &x);           //DI - PA25
GPIO_Init(GPIO_Pin_A26, &x);           //CLK - PA26
x.Func = GPIO_Reuse_Func0;
GPIO_Init(GPIO_Pin_A27, &x);           //CS - PA27

y.SPI_Mode = SPI_Mode_Master;          // SPI 模式
y.SPI_Df = SPI_FallSendRiseRec;        //设置数据格式
y.SPI_Freq = 80000;                     //波特率设置
y.SPI_DelayRec = Enable;                //接收延迟
y.SPI_DW = 7;                           //数据宽度设置
y.SPI_DelaySend = Disable;              //发送延迟
SPI_Init(SPI1, &y);                     //初始化 SPI1

SPI1_Enable();                           //使能 SPI1
}
```

第14章 FLASH存储器自编程IAP

14.1 寄存器结构

- ◆ 支持 FLASH 数据保护，进行 IAP 操作前需先进行解锁，去除相关寄存器的写保护。
- ◆ 支持程序存储器 FLASH 全擦除模式（仅在 SWD 调试时有效）和页擦除模式。
- ◆ 支持字编程模式，每个字包含 4 个字节。
- ◆ IAP 操作过程中可软件禁止全局中断；也可使能中断，将中断向量表和中断服务程序（ISR）复制到 SRAM，通过设置中断向量表重映射使能寄存器 SCU_TBLREMAPEN 和中断向量表偏移寄存器 SCU_TBLOFF 可调用 SRAM 中的中断服务程序（ISR）来响应中断。
- ◆ IAP 操作进入擦除或编程状态后，IAP 自动上锁，进入 FLASH 保护状态，下次 IAP 操作前需重新解锁。
- ◆ 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。
- ◆ 芯片内置 IAP 自编程硬件固化模块，在 IAP 自编程操作程序中可以调用这些自编程固化模块，以减少 SRAM 中的 IAP 操作代码量。

14.2 寄存器结构

IAP 的寄存器定义于文件 HR8P506.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 )
```

14.3 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。

```
/* Register UnLock */
#define IAP_RegUnLock() (IAP->UL.Word = 0x000000A5)
#define IAP_RegLock() (IAP->UL.Word = 0x0)

/* IAP Enable */
#define IAP_Enable() (IAP->CON.EN = 1)
#define IAP_Disable() (IAP->CON.EN = 0)
```

```
/* Flash IAP request */  
#define IAP_REQ() (IAP->CON.FLASH_REQ = 1)  
#define IAP_REQ_END() (IAP->CON.FLASH_REQ = 0)
```

14.4 库函数

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

14.4.1 IAP_Unlock函数

- ◆函数原型: `ErrorStatus IAP_Unlock(void);`
- ◆功能描述: IAP 解锁
- ◆输入参数: 无
- ◆返回值: 成功或失败

14.4.2 IAP_WriteEnd函数

- ◆函数原型: `ErrorStatus IAP_WriteEnd(void);`
- ◆功能描述: 写结束
- ◆输入参数: 无
- ◆返回值: 成功或失败

14.4.3 IAP_ErasePage函数

- ◆函数原型: `ErrorStatus IAP_ErasePage(uint8_t Page_Addr);`
- ◆功能描述: 页擦除
- ◆输入参数:
 Page_Addr: 页地址
- ◆返回值: 成功或失败

14.4.4 IAP_WriteCont函数

- ◆函数原型: `ErrorStatus IAP_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: 页内地址
 Page_addr: 页地址
 Data32: 数据
- ◆返回值: 成功或失败

14.4.5 IAP_WriteWord函数

- ◆函数原型: `ErrorStatus IAP_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: 页内地址

Page_addr: 页地址

Data32: 数据

◆返回值: 成功或失败

14.4.6 IAP_Read函数

◆函数原型: `ErrorStatus IAP_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: 读取数据的存放地址

Flash_Addr: 要读取的 Flash 地址

Len: 读取的字长度

◆返回值: 成功或失败

14.5 库函数应用示例

```
#define START_ADDR    0x00006400
#define PAGE_ADDR (START_ADDR / 1024)
uint32_t rbuf;
uint32_t wbuf;
uint32_t addr = 0;
IAP_Read(&rbuf, START_ADDR + addr, 1);    //读取 4 字节
IAP_WriteWord(addr, PAGE_ADDR, wbuf);    //写入 4 字节
IAP_ErasePage(PAGE_ADDR);                //页擦除
```


第15章 看门狗定时器WDT

15.1 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

15.2 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。

15.3 寄存器结构

WDT 的寄存器定义于文件 HR8P506.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 )
```

15.4 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。

```
#define WDT_RegUnLock()      (WDT->LOCK.Word = 1ACCE551)
#define WDT_RegLock()       (WDT->LOCK.Word = 0x0)

#define WDT_Enable()        (WDT->CON.EN = 1)
#define WDT_Disable()       (WDT->CON.EN = 0)

#define WDT_Clear()         (WDT->INTCLR = 0)

#define WDT_ITEnable()      (WDT->CON.IE = 1)
#define WDT_ITDisable()    (WDT->CON.IE = 0)

#define WDT_RstEnable()     (WDT->CON.RSTEN = 1)
#define WDT_RstDisable()   (WDT->CON.RSTEN = 0)
```

```
#define WDT_CLOCK_PCLK() (WDT->CON.CLKS = 0)
```

```
#define WDT_CLOCK_WDT() (WDT->CON.CLKS = 1)
```

15.5 库函数

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

15.5.1 WDT_Init函数

◆函数原型：void WDT_Init(WDT_InitStruType *WDT_InitStruct);

◆功能描述：WDT 初始化

◆输入参数：

WDT_InitStruct：初始化结构体指针

◆返回值：无

```
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_Clock;    /* 时钟选择 */
} WDT_InitStruType;

typedef enum {
    WDT_CLOCK_PCLK = 0x0,       /* PCLK */
    WDT_CLOCK_WDT = 0x1,        /* WDT 时钟源，32kHz */
} WDT_TYPE_CLKS;
```

15.5.2 WDT_SetReloadValue函数

◆函数原型：void WDT_SetReloadValue(uint32_t Value);

◆功能描述：设置 WDT 重载初值

◆输入参数：

Value：WDT 重载初值，32-bit 无符号整数

◆返回值：无

15.5.3 WDT_GetValue函数

◆函数原型：uint32_t WDT_GetValue(void);

◆功能描述：获取 WDT 当前计数值

◆输入参数：无

◆返回值：WDT 当前计数值

15.5.4 WDT_GetFlagStatus函数

◆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

15.6 库函数应用示例

```
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_CLOCK_WDT; //时钟源选择
    WDT_Init(&x);                //初始化 WDT
    WDT_Enable();                //使能 WDT
}
```

第16章 实时时钟RTC

16.1 寄存器结构

- ◆ 仅 POR 上电复位有效，支持程序写保护，有效避免系统干扰对时钟造成的影响
- ◆ 采用外部 32.768KHz 晶体振荡器作为 RTC 精确计时的时钟源；如果应用系统对 RTC 计时精度要求不高，还可选用内部 LRC 作为时钟源；如果应用系统将 RTC 作为普通计数器使用，还可选用 PCLK 或 PCLK 的 256 分频作为时钟源
- ◆ 可进行高精度数字校正，提供高精度计时
- ◆ 时钟调校提供两种时间精度，调校范围为 $\pm 384\text{ppm}$ （或 $\pm 128\text{ppm}$ ），可实现最大时间精度为 $\pm 1.5\text{ ppm}$ （或 $\pm 0.5\text{ppm}$ ）
- ◆ 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 5 个可编程定时中断
- ◆ 提供 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，有效期至 2099 年
- ◆ 12 小时和 24 小时模式设置可选
- ◆ 低功耗设计：工作电压 VDD=5.0V 时模块工作电流典型值为 0.5 μA

16.2 寄存器结构

RTC 的寄存器定义于文件 HR8P506.h。

typedef struct

```
{  
    __IO RTC_CON_Typedef CON;  
    __IO RTC_CAL_Typedef CAL;  
    __IO RTC_WA_Typedef WA;  
    __IO RTC_DA_Typedef DA;  
    __IO RTC_HMS_Typedef HMS;  
    __IO RTC_YMDW_Typedef YMDW;  
    __IO RTC_IE_Typedef IE;  
    __IO RTC_IF_Typedef IF;  
    __IO RTC_WP_Typedef WP;  
} RTC_TypeDef;
```

```
#define APB_BASE (0x40000000UL)  
#define RTC_BASE (APB_BASE + 0x01400)  
#define RTC ((RTC_TypeDef *) RTC_BASE )
```

16.3 宏定义

无

16.4 库函数

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

16.4.1 RTC_Init函数

- ◆函数原型: void RTC_Init(RTC_TYPE_CLKS CLKx,RTC_TYPE_TIME HOURx)
- ◆功能描述: 实时时钟初始化
- ◆输入参数:
 - CLKx:RTC 时钟源选择
 - HOURx: 12 小时/24 小时制选择
- ◆返回值: 无

16.4.2 RTC_ReadSecond函数

- ◆函数原型: uint32_t RTC_ReadSecond(void)
- ◆功能描述: 读当前时间的秒
- ◆输入参数:
 - CLKx:RTC 时钟源选择
 - HOURx: 12 小时/24 小时制选择
- ◆返回值: 秒

16.4.3 RTC_ReadMinute函数

- ◆函数原型: uint32_t RTC_ReadMinute(void)
- ◆功能描述: 读当前时间的分
- ◆输入参数: 无
- ◆返回值: 分

16.4.4 RTC_ReadHour函数

- ◆函数原型: uint32_t RTC_ReadHour(void)
- ◆功能描述: 读当前时间的小时
- ◆输入参数: 无
- ◆返回值: 小时

16.4.5 RTC_ReadDay函数

- ◆函数原型: uint32_t RTC_ReadDay(void)
- ◆功能描述: 读当前时间的日
- ◆输入参数: 无
- ◆返回值: 日

16.4.6 RTC_ReadMonth函数

- ◆函数原型: uint32_t RTC_ReadMonth(void)
- ◆功能描述: 读当前时间的月
- ◆输入参数: 无
- ◆返回值: 月

16.4.7 RTC_ReadYear函数

- ◆函数原型: uint32_t RTC_ReadYear(void)
- ◆功能描述: 读当前时间的年
- ◆输入参数: 无
- ◆返回值: 年

16.4.8 RTC_WriteSecond函数

- ◆函数原型: void RTC_WriteSecond(uint32_t second)
- ◆功能描述: 修改当前时间的秒
- ◆输入参数:
 - second: 秒, 有效范围是[0, 59]
- ◆返回值: 无

16.4.9 RTC_WriteMinute函数

- ◆函数原型: void RTC_WriteMinute(uint32_t minute)
- ◆功能描述: 修改当前时间的分钟
- ◆输入参数:
 - minute: 分钟, 有效范围是[0, 59]
- ◆返回值: 无

16.4.10 RTC_WriteHour函数

- ◆函数原型: void RTC_WriteHour(uint32_t hour)
- ◆功能描述: 修改当前时间的小时
- ◆输入参数:
 - hour: 小时, 有效范围是[0, 23]
- ◆返回值: 无

16.4.11 RTC_WriteDay函数

- ◆函数原型: void RTC_WriteDay(uint32_t day)
- ◆功能描述: 修改当前时间的日
- ◆输入参数:
 - day: 日, 有效范围是[1, 31]
- ◆返回值: 无

16.4.12 RTC_WriteMonth函数

- ◆函数原型: void RTC_WriteMonth(uint32_t month)
- ◆功能描述: 修改当前时间的月
- ◆输入参数:
 - month: 月, 有效范围是[1, 12]
- ◆返回值: 无

16. 4. 13 RTC_WriteYear函数

- ◆函数原型: void RTC_WriteYear(uint32_t year)
- ◆功能描述: 修改当前时间的年
- ◆输入参数:
 year: 年, 有效范围是[2000, 2099]
- ◆返回值: 无

16. 4. 14 RTC_InterruptEnable函数

- ◆函数原型: void RTC_InterruptEnable(RTC_Interrupt_Source src)
- ◆功能描述: 使能实时时钟的某些中断
- ◆输入参数:
 src: 实时时钟的中断源
- ◆返回值: 无

RTC 中断源选择:

```
typedef enum {  
    RTC_Interrupt_Source_Second = 0,           // 秒中断  
    RTC_Interrupt_Source_Minute = 1,           // 分中断  
    RTC_Interrupt_Source_Hour = 2,             // 小时中断  
    RTC_Interrupt_Source_Day = 3,              // 天中断  
    RTC_Interrupt_Source_Month = 4,            // 月中断  
} RTC_Interrupt_Source;
```

16. 4. 15 RTC_InterruptDisable函数

- ◆函数原型: void RTC_InterruptDisable(RTC_Interrupt_Source src)
- ◆功能描述: 禁能实时时钟的某些中断
- ◆输入参数:
 src: 实时时钟的中断源
- ◆返回值: 无

RTC 中断源选择:

```
typedef enum {  
    RTC_Interrupt_Source_Second = 0,           // 秒中断  
    RTC_Interrupt_Source_Minute = 1,           // 分中断  
    RTC_Interrupt_Source_Hour = 2,             // 小时中断  
    RTC_Interrupt_Source_Day = 3,              // 天中断  
    RTC_Interrupt_Source_Month = 4,            // 月中断  
} RTC_Interrupt_Source;
```

16. 4. 16 RTC_GetITFlag函数

- ◆函数原型: uint32_t RTC_GetITFlag(RTC_Interrupt_Source src)

- ◆功能描述：读取实时时钟的某个中断标志位
- ◆输入参数：
 - src: 实时时钟的中断源
- ◆返回值：
 - 0: 中断标志位是 0
 - 1: 中断标志位是 1

RTC 中断源选择:

```
typedef enum {  
    RTC_Interrupt_Source_Second = 0,      // 秒中断  
    RTC_Interrupt_Source_Minute = 1,      // 分中断  
    RTC_Interrupt_Source_Hour = 2,        // 小时中断  
    RTC_Interrupt_Source_Day = 3,         // 天中断  
    RTC_Interrupt_Source_Month = 4,       // 月中断  
} RTC_Interrupt_Source;
```

16. 4. 17 RTC_ClearAllITFlag函数

- ◆函数原型：void RTC_ClearAllITFlag(void)
- ◆功能描述：清除实时时钟的所有中断标志位
- ◆输入参数：无
- ◆返回值：无

16. 5 库函数应用示例

```
uint32_t RTC_Test(void)  
{  
    uint32_t sec, min, hour, day, month, year;  
    uint32_t result;  
  
    RTC_Init();  
    /*读取上电默认值 */  
    sec = RTC_ReadSecond();  
    min = RTC_ReadMinute();  
    hour = RTC_ReadHour();  
    day = RTC_ReadDay();  
    month = RTC_ReadMonth();  
    year = RTC_ReadYear();  
    /*写入新的值 */  
    RTC_WriteSecond(17);  
    sec = RTC_ReadSecond();  
    RTC_WriteMinute(5);  
    min = RTC_ReadMinute();  
    RTC_WriteHour(11);  
    hour = RTC_ReadHour();  
}
```



```
RTC_WriteDay(2);
day = RTC_ReadDay();
RTC_WriteMonth(12);
month = RTC_ReadMonth();
RTC_WriteYear(2015);
year = RTC_ReadYear();
/*读取新的值 */
if ((sec != 17) || (min != 5) || (hour != 11) || (day != 2) || (month != 12) || (year != 2015))
    result = 1;
else
    result = 0;

return result;
}
```

第17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。

17.1 UART波特率误差

误差可按照以下步骤计算：

计算 BRR 寄存器值

$$BRR = \text{INT} \left(\frac{F_{\text{pclk}}}{\text{Dbaud} \times n} - 1 \right)$$

若 $BRR > 2047$ ，则 $BRR = 2047$ 。

其中， F_{pclk} 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 $\text{UART_ClockSet} = \text{UART_Clock_1}$ ， $n=16$ ；若 $\text{UART_ClockSet} = \text{UART_Clock_2}$ ， $n=32$ ；若 $\text{UART_ClockSet} = \text{UART_Clock_3}$ ， $n=64$ 。

计算真实波特率

$$R_{\text{baud}} = \frac{F_{\text{pclk}}}{(BRR + 1) \times n}$$

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

计算误差

$$\text{误差} = \frac{R_{\text{baud}} - \text{Dbaud}}{\text{Dbaud}} \times 100\%$$

下表为 F_{pclk} 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 16-1 UART 波特率误差

17.2 IIC波特率误差

误差可按照以下步骤计算：

计算 TJP 寄存器值

$$TJP = \text{INT} \left(\frac{F_{\text{clk}}}{\text{Dbaud} \times n} - 1 \right)$$

若 $TJP > 255$ ，则 $TJP = 255$ 。

其中， F_{clk} 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 $\text{IIC_16XSamp} = \text{Disable}$ ， $n = 16$ ；若 $\text{IIC_16XSamp} = \text{Enable}$ ， $n = 24$ 。

计算真实波特率

$$R_{\text{baud}} = \frac{F_{\text{clk}}}{(TJP + 1) \times n}$$

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

计算误差

$$\text{误差} = \frac{R_{\text{baud}} - \text{Dbaud}}{\text{Dbaud}} \times 100\%$$

下表为 F_{clk} 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 16-2 IIC 波特率误差

17.3 SPI波特率误差

误差可按照以下步骤计算：

计算 CKS 寄存器值

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

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）。

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。

计算误差

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

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 16-3 SPI 波特率误差