

文档编号: AN_092

上海东软载波微电子有限公司

用户手册

ES8P508x 库函数

修订历史

版本	修订日期	修改概要
V1.0	2017-12-20	初版

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

内容目录

第 1 章	概述	10
1.1	关于本文档	10
1.2	芯片简介	10
1.3	芯片时钟树	14
1.4	文档规范	14
1.4.1	缩写	14
1.4.2	命名规则	15
1.4.3	数据类型	15
第 2 章	开始使用	17
2.1	文件结构	17
2.2	函数库的配置	17
2.2.1	printf 函数使用串口的选择	18
2.2.2	函数库的引用	18
2.3	中断函数	19
第 3 章	系统控制单元（SCU）	20
3.1	功能概述	20
3.2	特殊说明	20
3.3	寄存器结构	20
3.4	宏定义	21
3.5	库函数	26
3.5.1	函数 SCU_NMISelect	26
3.5.2	函数 SCU_GetPWRCFlagStatus	27
3.5.3	函数 SCU_ClearPWRCFlagBit	28
3.5.4	函数 SCU_GetLVDFFlagStatus	28
3.5.5	函数 SCU_SysClkSelect	29
3.5.6	函数 SCU_GetSysClk	29
3.5.7	函数 SCU_HRCReadyFlag	29
3.5.8	函数 SCU_XTALReadyFlag	29
3.5.9	函数 SCU_PLLReadyFlag	30
3.5.10	函数 SystemClockConfig	30
3.5.11	函数 DeviceClockAllEnable	30
3.5.12	函数 DeviceClockAllDisable	30
3.5.13	函数 SystemClockSelect	31
3.5.14	函数 PLLClock_Config	31
3.6	函数库应用示例	32
第 4 章	内核模块	33
4.1	功能概述	33
4.2	寄存器结构	33
4.3	宏定义	34
4.4	库函数	34

4.4.1	函数 NVIC_Init	35
4.4.2	函数 SCB_SystemLPConfig	36
4.4.3	函数 SCB_GetCpuID	36
4.4.4	函数 SysTick_Init.....	37
4.5	函数库应用示例	37
第 5 章	通用输入输出 (GPIO)	38
5.1	功能概述	38
5.2	特殊说明	38
5.3	寄存器结构	38
5.4	宏定义.....	40
5.5	库函数.....	41
5.5.1	函数 GPIO_Init.....	41
5.5.2	函数 GPIO_Write	44
5.5.3	函数 GPIO_Read	44
5.5.4	函数 GPIO_ReadBit.....	45
5.5.5	函数 GPIOA_SetBit.....	45
5.5.6	函数 GPIOA_ResetBit.....	45
5.5.7	函数 GPIOA_ToggleBit	45
5.5.8	函数 GPIOB_SetBit.....	45
5.5.9	函数 GPIOB_ResetBit.....	46
5.5.10	函数 GPIOB_ToggleBit	46
5.5.11	函数 GPIOA_SetDirection.....	46
5.5.12	函数 GPIOB_SetDirection.....	46
5.5.13	函数 PINT_Config	46
5.5.14	函数 PINT_GetITStatus	48
5.5.15	函数 PINT_ClearITPendingBit	48
5.5.16	函数 GPIO_SetFuncxRegFromPin	48
5.5.17	函数 GPIO_SetSingalTypeFromPin.....	49
5.5.18	函数 GPIO_SetDirRegFromPin	49
5.5.19	函数 GPIO_SetODERegFromPin	49
5.5.20	函数 GPIO_SetDSRegFromPin.....	49
5.5.21	函数 GPIO_SetPUERegFromPin	50
5.5.22	函数 GPIO_SetPDERegFromPin	50
5.6	函数库应用示例	50
第 6 章	定时器/计数器 (T16N/T32N)	52
6.1	功能概述	52
6.1.1	T16N	52
6.1.2	T32N	52
6.2	寄存器结构	53
6.3	宏定义.....	54
6.4	库函数.....	56
6.4.1	函数 T16Nx_BaseInit	56
6.4.2	函数 T32Nx_BaseInit	56
6.4.3	函数 T16Nx_CapInit.....	57

6.4.4	函数 T32Nx_CapInit.....	58
6.4.5	函数 T16Nx_MATxITConfig	59
6.4.6	函数 T32Nx_MATxITConfig	59
6.4.7	函数 T16Nx_MATxOutxConfig.....	59
6.4.8	函数 T32Nx_MATxOutxConfig.....	60
6.4.9	函数 T16Nx_ITConfig	60
6.4.10	函数 T32Nx_ITConfig	61
6.4.11	函数 T16Nx_PWMOutConfig.....	61
6.4.12	函数 T16Nx_PWMBK_Config.....	62
6.4.13	函数 T16Nx_TRG_Config.....	64
6.4.14	函数 T16Nx_GetPWMBKF	64
6.4.15	函数 T16Nx_ResetPWMBKF.....	64
6.4.16	函数 T16Nx_SetCNT	64
6.4.17	函数 T32Nx_SetCNT	65
6.4.18	函数 T16Nx_SetPRECNT.....	65
6.4.19	函数 T32Nx_SetPRECNT.....	65
6.4.20	函数 T16Nx_SetPREMAT	65
6.4.21	函数 T32Nx_SetPREMAT.....	66
6.4.22	函数 T16Nx_SetMATx.....	66
6.4.23	函数 T32Nx_SetMATx.....	66
6.4.24	函数 T16Nx_GetMATx	67
6.4.25	函数 T32Nx_GetMATx	67
6.4.26	函数 T16Nx_GetCNT	68
6.4.27	函数 T32Nx_GetCNT	68
6.4.28	函数 T16Nx_GetPRECNT	68
6.4.29	函数 T32Nx_GetPRECNT	68
6.4.30	函数 T16Nx_GetFlagStatus.....	68
6.4.31	函数 T32Nx_GetFlagStatus.....	69
6.4.32	函数 T16Nx_GetITStatus.....	69
6.4.33	函数 T32Nx_GetITStatus.....	69
6.4.34	函数 T16Nx_ClearIFPendingBit.....	69
6.4.35	函数 T32Nx_ClearIFPendingBit.....	70
6.5	函数库应用示例	70
第 7 章	模拟/数字转换器 (ADC)	71
7.1	功能概述	71
7.2	寄存器结构	71
7.3	宏定义.....	72
7.4	库函数.....	72
7.4.1	函数 ADC_Init	72
7.4.2	函数 ADC_Set_CH.....	75
7.4.3	函数 ADC_GetConvValue.....	75
7.4.4	函数 ADC_GetConvStatus.....	76
7.4.5	函数 ADC_ACPCConfig	76
7.4.6	函数 ADC_SampStart	77

7.4.7	函数 ADC_SampStop	77
7.4.8	函数 ADC_GetACPMeanValue	77
7.4.9	函数 ADC_GetACPMinValue	77
7.4.10	函数 ADC_GetACPMaxValue	77
7.4.11	函数 ADC_GetFlagStatus	78
7.4.12	函数 ADC_GetITStatus	78
7.4.13	函数 ADC_ClearIFStatus	78
7.4.14	函数 ADC_Reset	79
7.5	库函数应用示例	79
第8章	通用异步收发器 (UART)	81
8.1	功能概述	81
8.2	寄存器结构	81
8.3	宏定义	82
8.4	库函数	83
8.4.1	函数 UART_Init	83
8.4.2	函数 UART_ITConfig	84
8.4.3	函数 UART_TBIMConfig	85
8.4.4	函数 UART_RBIMConfig	85
8.4.5	函数 UART_Send	85
8.4.6	函数 UART_Rec	86
8.4.7	函数 UART_GetFlagStatus	86
8.4.8	函数 UART_GetITStatus	87
8.4.9	函数 UART_ClearIFPendingBit	87
8.4.10	函数 UART_printf	87
8.4.11	函数 fputc	88
8.4.12	函数 static char *itoa	88
8.5	函数库应用示例	88
第9章	IIC 串行总线	90
9.1	功能概述	90
9.2	寄存器结构	90
9.3	宏定义	91
9.4	库函数	92
9.4.1	函数 IIC_Init	92
9.4.2	函数 I2C_ITConfig	93
9.4.3	函数 I2C_SendAddress	93
9.4.4	函数 I2C_SetAddress	94
9.4.5	函数 I2C_RecModeConfig	94
9.4.6	函数 I2C_TBIMConfig	94
9.4.7	函数 I2C_RBIMConfig	95
9.4.8	函数 I2C_AckDelay	95
9.4.9	函数 I2C_TISConfig	96
9.4.10	函数 I2C_Send	96
9.4.11	函数 I2C_Rec	97
9.4.12	函数 I2C_GetRWMode	97

9. 4. 13	函数 I2C_GetTBStatus	97
9. 4. 14	函数 I2C_GetFlagStatus	97
9. 4. 15	函数 I2C_GetITStatus	98
9. 4. 16	函数 I2C_ClearITPendingBit	98
9. 5	函数库应用示例	98
第 10 章	SPI 串行总线	100
10. 1	功能概述	100
10. 2	寄存器结构	100
10. 3	宏定义	100
10. 4	库函数	101
10. 4. 1	函数 SPI_Init	101
10. 4. 2	函数 SPI_ITConfig	102
10. 4. 3	函数 SPI_DataFormatConfig	102
10. 4. 4	函数 SPI_Send	102
10. 4. 5	函数 SPI_Rec	103
10. 4. 6	函数 SPI_TBIMConfig	103
10. 4. 7	函数 SPI_RBIMConfig	103
10. 4. 8	函数 SPI_GetFlagStatus	103
10. 4. 9	函数 SPI_GetITStatus	104
10. 4. 10	函数 SPI_ClearITPendingBit	104
10. 5	函数库应用示例	105
第 11 章	FLASH 存储器自编程 (IAP)	106
11. 1	功能概述	106
11. 2	寄存器结构	106
11. 3	宏定义	106
11. 4	库函数	107
11. 4. 1	函数 Flashlap_Close_WPROT	107
11. 4. 2	函数 Flashlap_Open_WPROT	107
11. 4. 3	函数 Flashlap_CloseAll_WPROT	107
11. 4. 4	函数 Flashlap_OpenAll_WPROT	107
11. 4. 5	函数 Flashlap_Unlock	108
11. 4. 6	函数 Flashlap_WriteEnd	108
11. 4. 7	函数 Flashlap_ErasePage	108
11. 4. 8	函数 Flashlap_WriteCont	108
11. 4. 9	函数 Flashlap_WriteWord	108
11. 4. 10	函数 Flash_Read	109
11. 5	函数库应用示例	109
第 12 章	硬件独立看门狗 (IWDG)	110
12. 1	功能概述	110
12. 2	特殊说明	110
12. 3	寄存器结构	110
12. 4	宏定义	110
12. 5	库函数	111
12. 5. 1	函数 IWDG_Init	111

12.5.2	函数 IWDT_SetReloadValue.....	112
12.5.3	函数 IWDT_GetValue.....	112
12.5.4	函数 IWDT_GetFlagStatus	112
12.5.5	函数 IWDT_GetITStatus	112
12.6	函数库应用示例	113
第 13 章	窗口看门狗 (WWDT)	114
13.1	功能概述	114
13.2	特殊说明	114
13.3	寄存器结构	114
13.4	宏定义.....	114
13.5	库函数.....	115
13.5.1	函数 WWDT_Init	115
13.5.2	函数 WWDT_SetReloadValue	116
13.5.3	函数 WWDT_GetValue	116
13.5.4	函数 WWDT_GetFlagStatus	117
13.5.5	函数 Status WWDT_GetITStatus	117
13.6	函数库应用示例	117
第 14 章	循环冗余校验 (CRC)	118
14.1	功能概述	118
14.2	特殊说明	118
14.3	寄存器结构	118
14.4	宏定义.....	118
14.5	库函数.....	119
14.5.1	函数 CRC_getTypeValue	119
14.5.2	函数 CRC_EmptayCheck	120
14.5.3	函数 CRC_FlashVerify	120
14.5.4	函数 CRC_UserCal.....	121
14.5.5	函数 CRC_CheckReset	121
14.6	函数库应用示例	121
第 15 章	循环冗余校验 (CRC)	122
15.1	功能概述	122
15.2	寄存器结构	122
15.3	宏定义.....	122
15.4	库函数.....	122
15.4.1	函数 AES_Init.....	123
15.4.2	函数 AES_WriteKey	123
15.4.3	函数 AES_ReadKey.....	123
15.4.4	函数 AES_WriteData.....	123
15.4.5	函数 AES_ReadData	124
15.4.6	函数 AES_ITConfig.....	124
15.4.7	函数 AES_GetFlagStatus	124
15.4.8	函数 AES_ClearITPendingBit	124
15.4.9	函数 AES_GetDoneStatus.....	125
15.4.10	函数 AES_Reset	125

15.5	函数库应用示例	125
第 16 章	实时时钟 (RTC)	127
16.1	功能概述	127
16.2	寄存器结构	127
16.3	宏定义	128
16.4	库函数	128
16.4.1	函数 RTC_Init	128
16.4.2	函数 RTC_StartRead	128
16.4.3	函数 RTC_ReadHourmode	129
16.4.4	函数 RTC_ReadSecond	129
16.4.5	函数 RTC_ReadMinute	129
16.4.6	函数 RTC_ReadHour	129
16.4.7	函数 RTC_ReadDay	129
16.4.8	函数 RTC_ReadMonth	130
16.4.9	函数 RTC_ReadYear	130
16.4.10	函数 RTC_ReadWeek	130
16.4.11	函数 RTC_Start Write	130
16.4.12	函数 RTC_WriteSecond	130
16.4.13	函数 RTC_WriteMinute	131
16.4.14	函数 RTC_WriteHour	131
16.4.15	函数 RTC_WriteDay	131
16.4.16	函数 RTC_Write Month	131
16.4.17	函数 RTC_WriteYear	131
16.4.18	函数 RTC_WriteWeek	132
16.4.19	函数 RTC_ReadWeekAlarmMinute	132
16.4.20	函数 RTC_ReadWeekAlarmHour	132
16.4.21	函数 RTC_ReadWeekAlarmWeek	132
16.4.22	函数 RTC_ReadDayAlarmMinute	132
16.4.23	函数 RTC_ReadDayAlarmHour	133
16.4.24	函数 RTC_WriteWeekAlarmMinute	133
16.4.25	函数 RTC_WriteWeekAlarmHour	133
16.4.26	函数 RTC_WriteWeekAlarmWeek	133
16.4.27	函数 RTC_WriteDayAlarmMinute	133
16.4.28	函数 RTC_WriteDayAlarmHour	134
16.4.29	函数 RTC_InterruptEnable	134
16.4.30	函数 RTC_InterruptDisable	134
16.4.31	函数 RTC_GetITStatus	134
16.4.32	函数 RTC_GetFlagStatus	135
16.4.33	函数 RTC_ClearAllITFlag	135
16.5	函数库应用示例	135
第 17 章	波特率误差	136
17.1	UART 波特率误差	136
17.2	IIC 波特率误差	137
17.3	SPI 波特率误差	138

第1章 概述

1.1 关于本文档

本文档是 ES8P508x 系列芯片固件函数库的应用笔记。函数库提供了芯片内资源与外设的驱动接口，用户使用函数库进行软件开发，可避免直接对芯片内寄存器的操作，从而缩短开发周期。本文档会对函数库中的每一个驱动接口进行描述，某些接口还会附以示例代码。

1.2 芯片简介

该产品是一款高集成度的通用 MCU 芯片，内部集成 32 位 ARM Cortex-M0 CPU 内核。内部集成 16 位和 32 位定时器/计数器，实时时钟模块 RTC，带红外发送调制功能的 UART 模块，SPI 和 I2C 通信模块，128 位 AES 以及用于系统电源监测的 LVD，分辨率最高为 12 位的可配置 ADC 模块等资源和外设。

◆ 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，400KHz~48MHz
- ◇ 工作电流：I_{vdd} = 4.5mA(@内部 HRC 20MHz，典型值)
- ◇ 待机电流：I_{vdd} = 5uA（常温，典型值）

◆ 封装

- ◇ LQFP48/LQFP44 封装（ES8P5088）
- ◇ SOP32/LQFP32 封装（ES8P5086）

◆ 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 20MHz RC 振荡器（HRC）可配置为系统时钟源，出厂前已校准（全温度，全电压范围内频率精度为±2%）。
- ◇ 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
- ◇ 系统上电默认主时钟为 20MHz HRC 时钟

◆ 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)
- ◇ 支持一组 SWD 调试接口
- ◇ 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
- ◇ 支持单周期 32 位乘法器

◆ 窗口看门狗 WWDT

- ◇ 时钟源可选择, 可用于检测软件的过早或过晚异常
- ◇ 安全可靠, 一旦使能, 只能通过复位关断
- ◇ 可设定喂狗窗口, 喂狗窗口外喂狗将产生复位

◆ 存储器

- ◇ 最大 128K 字节 FLASH 存储器
 - 72K 字节 FLASH 存储器 (ES8P5086)
 - 128K 字节 FLASH 存储器 (ES8P5088)
 - 支持 ISP 在线串行编程
 - 支持一组 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 全加密的编程代码加密保护
- ◇ 支持 8K 字节 Boot Flash
 - 通过芯片配置字设置从 Boot Flash 或主程序区启动
- ◇ 最大 24K 字节 SRAM 存储器
 - 16K 字节 SRAM 存储器 (ES8P5086)
 - 24K 字节 SRAM 存储器 (ES8P5088)
 - SRAM 存储空间及外设寄存器地址空间支持位带 (BIT BAND) 扩展

◆ 通用 CRC16/32

- ◇ 支持 Flash 数据完整性检查
- ◇ 支持数据通信 CRC 校验
- ◇ 可设定需进行 CRC 校验的 Flash 数据块的起始地址和大小

◆ AES 加密

- ◇ 128 位 AES, 支持数据加密/解密

◆ I/O 端口

- ◇ 最多 45 个双向 I/O 端口 (ES8P5088)
 - PA 端口 (PA0~PA31; 对 PA6 和 PA19, 不同型号的芯片只支持其中一个端口)
 - PB 端口 (PB0~PB13)
- ◇ 最多 29 个双向 I/O 端口 (ES8P5086)

- PA 端口 (PA2~PA5, PA7~PA12, PA17~PA22, PA27~PA31) (ES8P5086FJSK)
- PA 端口 (PA0~PA8, PA10~PA18, PA24~PA25) (ES8P5086FJLK)
- PB 端口 (PB0~PB7) (ES8P5086FJSK)
- PB 端口 (PB0~PB7, PB11) (ES8P5086FJLK)

- ◇ 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, 时钟源可选择

◆ UART 通信接口

- ◇ 支持 6 路 UART 通信接口 UART0/UART1/UART2/UART3/UART4/UART5
- ◇ 支持全/半双工异步通信模式
- ◇ 支持单线半双工异步通信模式
- ◇ 支持传输波特率可配置
- ◇ 支持 4 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
- ◇ 接收端口支持红外唤醒功能
- ◇ 支持 UART 输入输出通讯端口极性可配置

◆ I2C 通信接口

- ◇ 支持 1 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 1 路通信接口 SPI0
- ◇ 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 模拟数字转换器

- ◇ 支持 8/10/12 位分辨率，有效精度为 11 位
- ◇ 支持外部最多 15 通道模拟输入端
- ◇ 支持一路内部 1/4 VDD（或 1/3 VDD）通道输入
- ◇ 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

◆ 大电流驱动端口

- ◇ 8 个 40mA 大电流（灌电流）驱动口（PA6~PA13）
- ◇ 可用于驱动 1~8 个 8 段式共阳极数码管

◆ 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，有效期到 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 0.5 μA

1.3 芯片时钟树

ES8P508x 具有丰富的时钟及配置系统，详见下图：

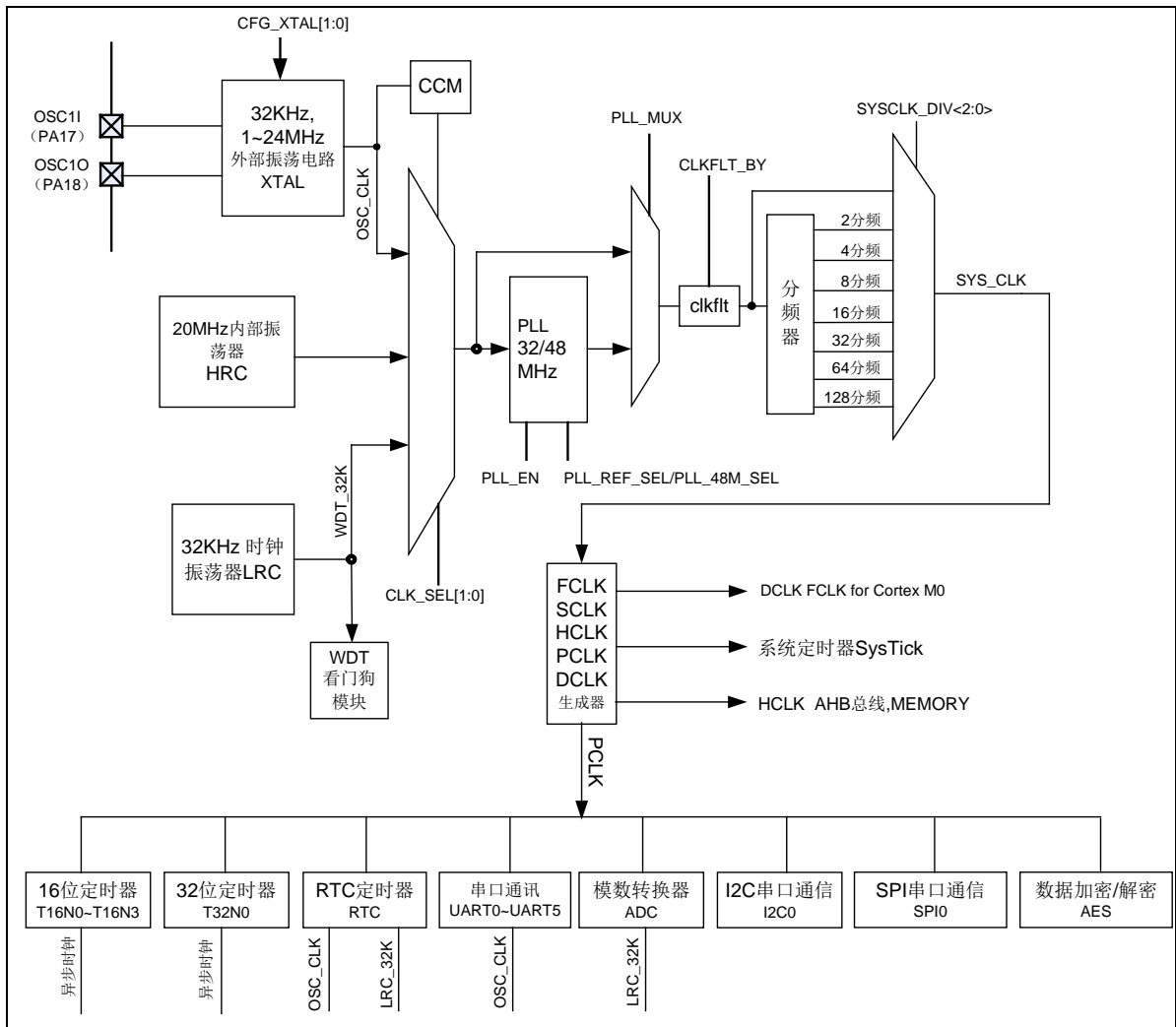


图 1-1 时钟树

1.4 文档规范

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

1.4.1 缩写

缩写	含义	缩写	含义
Flash	闪存存储器	IIC/I2C	集成电路总线
IAP	应用中自编程	SCU	系统控制单元
ADC	模数转换器	RTC	实时时钟
CRC	循环冗余校验	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	AES	数据加密/解密
IWDG	硬件独立看门狗	WWDT	窗口看门狗

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 = 0,
    ENABLE = !DISABLE
} 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。

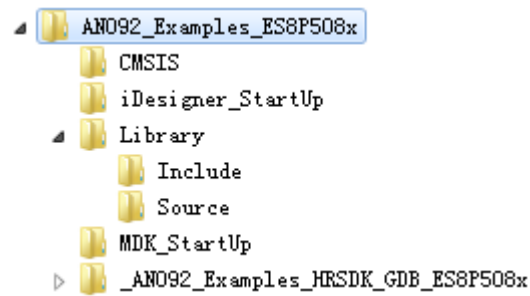


图 2-1 函数库文件夹结构

◆ 文件夹 CMSIS

该文件夹下存放 ARM 内核头文件 `core_cm0.h`，同时也存放了芯片的头文件 `ES8P508x.h`。

◆ 文件夹 iDesigner_StartUp

该文件夹下存放芯片的 iDesigner 启动文件 `ES8P508x_startup.S`。

◆ 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 MDK_StartUp

该文件夹下存放芯片的启动文件 `startup_ES8P508x.s`。

◆ 文件夹 _AN092_Examples_HRSDK_GDB_ES8P508x

该文件夹下存放基于 HRSDK_GDB 开发板的演示程序，包含 ADC、Flash、IAP、IIC、SPI、UART 等多个例程，同时包含可以使用学习子板单独演示的例程，包含 AES、CRC、PLL 等多个例程，每个例程下又包含以下文件夹：

- ◇ 文件夹 APP：存放用户程序的源代码及头文件。
- ◇ 文件夹 PlatForm：该文件夹下存放两个文件：`irqhandler.c` 与 `irqhandler.h`。文件内定义了相关的中断处理函数。
- ◇ 文件夹 MDK_Project：该文件夹下存放与开发环境相关的工程文件以及文件夹 Obj。
文件夹 Obj：存放编译器生成的编译、链接、列表及二进制文件等

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_ES8P508.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

在 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 UART2, 则定义 __PRINTF_USE_UART2__。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_wdt.h"
#include "lib_flashiap.h"
#include "lib_gpio.h"
#include "lib_printf.h"
#include "lib_iap.h"
#include "lib_crc.h"

#include "lib_aes.h"
```

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

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

2.3 中断函数

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

函数名	描述
PINT0_IRQHandler	外部中断 0
PINT1_IRQHandler	外部中断 1
PINT2_IRQHandler	外部中断 2
PINT3_IRQHandler	外部中断 3
PINT4_IRQHandler	外部中断 4
PINT5_IRQHandler	外部中断 5
PINT6_IRQHandler	外部中断 6
PINT7_IRQHandler	外部中断 7
T16N0_IRQHandler	16 位定时器 0 中断
T16N1_IRQHandler	16 位定时器 1 中断
T16N2_IRQHandler	16 位定时器 2 中断
T16N3_IRQHandler	16 位定时器 3 中断
T32N0_IRQHandler	32 位定时器 0 中断
IWDT_IRQHandler	独立看门狗中断
WWDT_IRQHandler	窗口看门狗中断
CCM_IRQHandler	停振检测中断
PLK_IRQHandler	PLL 失锁中断
LVD_IRQHandler	低电压检测中断
KINT_IRQHandler	外部按键输入中断
RTC_IRQHandler	实时时钟中断
ADC_IRQHandler	模数转换中断
AES_IRQHandler	AES 加解密中断
UART0_IRQHandler	串口 0 中断
UART1_IRQHandler	串口 1 中断
UART2_IRQHandler	串口 2 中断
UART3_IRQHandler	串口 3 中断
UART4_IRQHandler	串口 4 中断
UART5_IRQHandler	串口 5 中断
SPI0_IRQHandler	SPI0 中断
I2C0_IRQHandler	IIC0 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 20MHz RC 振荡器可配置为系统时钟源
- ◆ 内部 32KHz RC 振荡器可配置为系统时钟源
- ◆ 支持 PLL 时钟，时钟源可选择，最大可倍频至 48MHz，可配置为系统时钟源

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 ES8P508x.h。

```
typedef struct
{
    __IO SCU_PROT_Typedef PROT;
    __IO SCU_NMICON_Typedef NMICON;
    __IO SCU_PWRC_Typedef PWRC;
    __IO SCU_FAULTFLAG_Typedef FAULTFLAG;
    __IO SCU_WAKEUPTIME_Typedef WAKEUPTIME;
    __IO SCU_MRSTN_SOFT_Typedef MRSTN_SOFT;
    __IO SCU_DBGHALT_Typedef DBGHALT;
    uint32_t RESERVED0 ;
    __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;
```

```
uint32_t RESERVED1[3];

__IO SCU_SCLKEN0_Typedef SCLKEN0;
__IO SCU_SCLKEN1_Typedef SCLKEN1;
__IO SCU_PCLKEN0_Typedef PCLKEN0;
__IO SCU_PCLKEN1_Typedef PCLKEN1;
__IO SCU_PRSTEN0_Typedef PRSTEN0;
__IO SCU_PRSTEN1_Typedef PRSTEN1;
__IO SCU_TIMEREN_Typedef TIMEREN;
__IO SCU_TIMERDIS_Typedef TIMERDIS;
__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_2V0() (SCU->LVDCON.VS = 0x0)
```

```
#define SCU_LVDVS_2V1() (SCU->LVDCON.VS = 0x1)
```

```
#define SCU_LVDVS_2V2() (SCU->LVDCON.VS = 0x2)
```

```
#define SCU_LVDVS_2V4() (SCU->LVDCON.VS = 0x3)
```

```
#define SCU_LVDVS_2V6() (SCU->LVDCON.VS = 0x4)
```

```
#define SCU_LVDVS_2V8() (SCU->LVDCON.VS = 0x5)
```

```
#define SCU_LVDVS_3V0() (SCU->LVDCON.VS = 0x6)
```

```
#define SCU_LVDVS_3V6() (SCU->LVDCON.VS = 0x7)
```

```
#define SCU_LVDVS_4V() (SCU->LVDCON.VS = 0x8)
```

```
#define SCU_LVDVS_4V6() (SCU->LVDCON.VS = 0x9)
```

```
#define SCU_LVDVS_2V3() (SCU->LVDCON.VS = 0xA)
```

```
#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.LVDIF = 1)
```

```
/* LVD 中断产生模式选择 */
```

```
#define SCU_LVDIFS_Rise() (SCU->LVDCON.IFS = 0x0)
```

```
#define SCU_LVDIFS_Fall() (SCU->LVDCON.IFS = 0x1)
```

```
#define SCU_LVDIFS_High() (SCU->LVDCON.IFS = 0x2)
```

```
#define SCU_LVDIFS_Low() (SCU->LVDCON.IFS = 0x3)
```

```
#define SCU_LVDIFS_Change() (SCU->LVDCON.IFS = 0x4)
```

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

```
/* HRC 使能控制 (内部 20Mhz) */
```

```
#define SCU_HRC_Enable() (SCU->SCLKEN1.HRC_EN = 1)
#define SCU_HRC_Disable() (SCU->SCLKEN1.HRC_EN = 0)
```

```
/* XTAL 使能控制 */
```

```
#define SCU_XTAL_Enable() (SCU->SCLKEN1.XTAL_EN = 1)
#define SCU_XTAL_Disable() (SCU->SCLKEN1.XTAL_EN = 0)
```

```
/* PLL 模式使能控制 */
```

```
#define SCU_PLL_Enable() (SCU->SCLKEN1.PLL_EN = 1)
#define SCU_PLL_Disable() (SCU->SCLKEN1.PLL_EN = 0)
```

```
/*-----外设时钟控制-----*/

/* SCU 时钟使能控制 */

#define SCU_SCUCLK_Enable() (SCU->PCLKEN0.SCU_EN = 1)
#define SCU_SCUCLK_Disable() (SCU->PCLKEN0.SCU_EN = 0)

/* GPIO 时钟使能控制 */

#define SCU_GPIOCLK_Enable() (SCU->PCLKEN0.GPIO_EN = 1)
#define SCU_GPIOCLK_Disable() (SCU->PCLKEN0.GPIO_EN = 0)

/* FLASH IAP 时钟使能控制 */

#define SCU_IAPCLK_Enable() (SCU->PCLKEN0.IAP_EN = 1)
#define SCU_IAPCLK_Disable() (SCU->PCLKEN0.IAP_EN = 0)

/* CRC 时钟使能控制 */

#define SCU_CRCCLK_Enable() (SCU->PCLKEN0.CRC_EN = 1)
#define SCU_CRCCLK_Disable() (SCU->PCLKEN0.CRC_EN = 0)

/* ADC 时钟使能控制 */

#define SCU_ADCCLK_Enable() (SCU->PCLKEN0.ADC_EN = 1)
#define SCU_ADCCLK_Disable() (SCU->PCLKEN0.ADC_EN = 0)

/* RTC 时钟使能控制 */

#define SCU_RTCCLK_Enable() (SCU->PCLKEN0.RTC_EN = 1)
#define SCU_RTCCLK_Disable() (SCU->PCLKEN0.RTC_EN = 0)

/* IWDG 时钟使能控制 */

#define SCU_IWDGCLK_Enable() (SCU->PCLKEN0.IWDG_EN = 1)
#define SCU_IWDGCLK_Disable() (SCU->PCLKEN0.IWDG_EN = 0)

/* WWDG 时钟使能控制 */

#define SCU_WWDGCLK_Enable() (SCU->PCLKEN0.WWDG_EN = 1)
#define SCU_WWDGCLK_Disable() (SCU->PCLKEN0.WWDG_EN = 0)

/* AES 时钟使能控制 */

#define SCU_AESCLK_Enable() (SCU->PCLKEN0.AES_EN = 1)
#define SCU_AESCLK_Disable() (SCU->PCLKEN0.AES_EN = 0)
```



```
/* T16N0 时钟使能控制 */
```

```
#define SCU_T16N0CLK_Enable() (SCU->PCLKEN1.T16N0_EN = 1)
```

```
#define SCU_T16N0CLK_Disable() (SCU->PCLKEN1.T16N0_EN = 0)
```

```
/* T16N1 时钟使能控制 */
```

```
#define SCU_T16N1CLK_Enable() (SCU->PCLKEN1.T16N1_EN = 1)
```

```
#define SCU_T16N1CLK_Disable() (SCU->PCLKEN1.T16N1_EN = 0)
```

```
/* T16N2 时钟使能控制 */
```

```
#define SCU_T16N2CLK_Enable() (SCU->PCLKEN1.T16N2_EN = 1)
```

```
#define SCU_T16N2CLK_Disable() (SCU->PCLKEN1.T16N2_EN = 0)
```

```
/* T16N3 时钟使能控制 */
```

```
#define SCU_T16N3CLK_Enable() (SCU->PCLKEN1.T16N3_EN = 1)
```

```
#define SCU_T16N3CLK_Disable() (SCU->PCLKEN1.T16N3_EN = 0)
```

```
/* T32N0 时钟使能控制 */
```

```
#define SCU_T32N0CLK_Enable() (SCU->PCLKEN1.T32N0_EN = 1)
```

```
#define SCU_T32N0CLK_Disable() (SCU->PCLKEN1.T32N0_EN = 0)
```

```
/* UART0 时钟使能控制 */
```

```
#define SCU_UART0CLK_Enable() (SCU->PCLKEN1.UART0_EN = 1)
```

```
#define SCU_UART0CLK_Disable() (SCU->PCLKEN1.UART0_EN = 0)
```

```
/* UART1 时钟使能控制 */
```

```
#define SCU_UART1CLK_Enable() (SCU->PCLKEN1.UART1_EN = 1)
```

```
#define SCU_UART1CLK_Disable() (SCU->PCLKEN1.UART1_EN = 0)
```

```
/* UART2 时钟使能控制 */
```

```
#define SCU_UART2CLK_Enable() (SCU->PCLKEN1.UART2_EN = 1)
```

```
#define SCU_UART2CLK_Disable() (SCU->PCLKEN1.UART2_EN = 0)
```

```
/* UART3 时钟使能控制 */
```

```
#define SCU_UART3CLK_Enable() (SCU->PCLKEN1.UART3_EN = 1)
```

```
#define SCU_UART3CLK_Disable() (SCU->PCLKEN1.UART3_EN = 0)
```

```
/* UART4 时钟使能控制 */  
  
#define SCU_UART4CLK_Enable() (SCU->PCLKEN1.UART4_EN = 1)  
#define SCU_UART4CLK_Disable() (SCU->PCLKEN1.UART4_EN = 0)  
  
/* UART5 时钟使能控制 */  
  
#define SCU_UART5CLK_Enable() (SCU->PCLKEN1.UART5_EN = 1)  
#define SCU_UART5CLK_Disable() (SCU->PCLKEN1.UART5_EN = 0)  
  
/* SPI0 时钟使能控制 */  
  
#define SCU_SPI0CLK_Enable() (SCU->PCLKEN1.SPI0_EN = 1)  
#define SCU_SPI0CLK_Disable() (SCU->PCLKEN1.SPI0_EN = 0)  
  
/* IIC0 时钟使能控制 */  
  
#define SCU_IIC0CLK_Enable() (SCU->PCLKEN1.I2C0_EN = 1)  
#define SCU_IIC0CLK_Disable() (SCU->PCLKEN1.I2C0_EN = 0)  
  
/* 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->TBLOFFS.TBLOFF = (uint32_t)x)
```

3.5 库函数

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

3.5.1 函数 SCU_NMISelect

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

不可屏蔽中断枚举类型 SCU_TYPE_NMICS:

枚举元素	数值	描述
SCU_PINT0_IRQn	0	外部中断 0
SCU_PINT1_IRQn	1	外部中断 1
SCU_PINT2_IRQn	2	外部中断 2
SCU_PINT3_IRQn	3	外部中断 3
SCU_PINT4_IRQn	4	外部中断 4
SCU_PINT5_IRQn	5	外部中断 5
SCU_PINT6_IRQn	6	外部中断 6
SCU_PINT7_IRQn	7	外部中断 7
SCU_T16N0_IRQn	8	定时器中断: T16N0
SCU_T16N1_IRQn	9	定时器中断: T16N1
SCU_T16N2_IRQn	10	定时器中断: T16N2
SCU_T16N3_IRQn	11	定时器中断: T16N3
SCU_T32N0_IRQn	12	定时器中断: T32N0
SCU_IWDT_IRQn	14	独立看门狗中断
SCU_WWDT_IRQn	15	窗口看门狗中断
SCU_CCM_IRQn	16	停振检测中断
SCU_PLK_IRQn	17	PLL 失锁中断
SCU_LVD_IRQn	18	低电压检测中断
SCU_KINT_IRQn	19	外部按键输入中断
SCU_RTC_IRQn	20	实时时钟中断
SCU_ADC_IRQn	21	模数转换中断
SCU_AES_IRQn	23	AES 加解密中断
SCU_UART0_IRQn	24	UART0 中断
SCU_UART1_IRQn	25	UART1 中断
SCU_UART2_IRQn	26	UART2 中断
SCU_UART3_IRQn	27	UART3 中断
SCU_UART4_IRQn	28	UART4 中断
SCU_UART5_IRQn	29	UART5 中断
SCU_SPI0_IRQn	30	SPI0 中断
SCU_I2C0_IRQn	31	I2C0 中断

表 3-1 SCU_TYPE_NMICS

3.5.2 函数 SCU_GetPWRCFlagStatus

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

PWRC 寄存器标志位枚举类型 SCU_TYPE_PWRC:

枚举元素	数值	描述
SCU_PWRC_PORF	0x00001	5V POR 复位标志
SCU_PWRC_RRCF	0x00002	1.5V POR 复位标志
SCU_PWRC_PORRSTF	0x00004	POR 总复位标志
SCU_PWRC_BORF	0x00008	BOR 总复位标志
SCU_PWRC_WWDTRSTF	0x00010	WWDT 复位标志
SCU_PWRC_IWDTRSTF	0x00020	IWDT 复位标志
SCU_PWRC_MRSTF	0x00040	MRSTn 复位标志
SCU_PWRC_SOFTTRSTF	0x00080	软件复位标志

表 3-2 SCU_TYPE_PWRC

3.5.3 函数SCU_ClearPWRCFlagBit

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

3.5.4 函数SCU_GetLVDFlagStatus

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

LVD 寄存器标志位枚举类型 SCU_TYPE_LVD0CON:

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

表 3-3 SCU_TYPE_LVD0CON

3.5.5 函数 SCU_SysClkSelect

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

时钟源枚举类型 SCU_TYPE_SYSCCLK:

枚举元素	数值	描述
SCU_SysClk_HRC	0x0	内部 20MHZ RC 时钟
SCU_SysClk_XTAL	0x1	外部 2-20MHz 晶振时钟
SCU_SysClk_PLL	0x2	PLL 锁相环倍频时钟

表 3-4 SCU_TYPE_SYSCCLK

3.5.6 函数 SCU_GetSysClk

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

3.5.7 函数 SCU_HRCReadyFlag

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

3.5.8 函数 SCU_XTALReadyFlag

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

3.5.9 函数SCU_PLLReadyFlag

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

3.5.10 函数SystemClockConfig

- ◆ 函数原型: void SystemClockConfig(void)
- ◆ 功能描述: 系统时钟配置: 内部时钟 20MHz
- ◆ 输入参数: 无
- ◆ 返回值: 无

注意: 执行此函数前无需解除写保护, 且该函数执行后不会改变写保护的状态。

3.5.11 函数DeviceClockAllEnable

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

注意: 执行此函数前无需解除写保护, 且该函数执行后不会改变写保护的状态。

3.5.12 函数DeviceClockAllDisable

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

注意: 执行此函数前无需解除写保护, 且该函数执行后不会改变写保护的状态。

3.5.13 函数SystemClockSelect

- ◆ 函数原型: void SystemClockSelect(SCU_TYPE_SYSCLK SYSCLKx , SCU_TYPE_CLK_SEL CLK_SEL)
- ◆ 功能描述: 系统时钟选择
- ◆ 输入参数: SYSCLKx 系统时钟源选择, 详见表 3-4;
CLK_SEL 原始时钟选择 详见表 3-5
- ◆ 输出参数: 无
- ◆ 返回值: 无

注意: 执行此函数前无需解除写保护, 且该函数执行后不会改变写保护的状态。

系统时钟源选择枚举类型 SCU_TYPE_SYSCLK:

枚举元素	数值	描述
SCU_SysClk_HRC	0x0	内部 20MHz RC 时钟
SCU_SysClk_XTAL	0x1	外部晶振主时钟
SCU_SysClk_PLL	0x2	PLL 时钟

表 3-4 SCU_TYPE_SYSCLK

原始时钟源选择枚举类型 SCU_TYPE_CLK_SEL:

枚举元素	数值	描述
CLK_SEL_HRC	0x0	HRC 20M
CLK_SEL_LRC	0x1	LRC 32KHz
CLK_SEL_XTAL	0x2	外部晶振 XTAL

表 3-5 SCU_TYPE_CLK_SEL

3.5.14 函数PLLClock_Config

- ◆ 函数原型: PLLClock_Config(TYPE_FUNCEN pll_en , SCU_PLL_Origin pll_origin ,SCU_PLL_Out pll_out,TYPE_FUNCEN sys_pll)
- ◆ 功能描述: 系统时钟选择
- ◆ 输入参数: pll_en 是否开启 PLL, pll_origin: PLL 时钟源选择, 详见表 3-6, pll_out: PLL 输出频率选择, 详见表 3-7, sys_pll: 系统是否使用 PLL 时钟
- ◆ 输出参数: 无
- ◆ 返回值: 无

注意: 执行此函数前无需解除写保护, 且该函数执行后不会改变写保护的状态。

PLL 时钟源选择枚举类型 SCU_PLL_Origin:

枚举元素	数值	描述
SCU_PLL_HRC	0x0	PLL 时钟源 HRC
SCU_PLL_LRC	0x2	PLL 时钟源 LRC
SCU_PLL_XTAL_32K	0x3	PLL 时钟源 XTAL
SCU_PLL_XTAL_4M	0x4	PLL 时钟源 XTAL
SCU_PLL_XTAL_8M	0x5	PLL 时钟源 XTAL
SCU_PLL_XTAL_16M	0x6	PLL 时钟源 XTAL
SCU_PLL_XTAL_20M	0x7	PLL 时钟源 XTAL

表 3-6 SCU_PLL_Origin

PLL 输出频率选择枚举类型 SCU_PLL_Out:

枚举元素	数值	描述
SCU_PLL_32M	0x0	PLL 时钟输出为 32MHz
SCU_PLL_48M	0x1	PLL 时钟输出为 48MHz

表 3-7 SCU_PLL_Out

3.6 函数库应用示例

```

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

```


第4章 内核模块

4.1 功能概述

内核模块包括以下三个部分：

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

4.2 寄存器结构

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

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

typedef struct
{
    /* Offset: 0x000 (R/W)  Interrupt Set Enable Register */
    __IO uint32_t ISER[1];
    uint32_t RESERVED0[31];
    /* Offset: 0x080 (R/W)  Interrupt Clear Enable Register */
    __IO uint32_t ICER[1];
    uint32_t RESERVED1[31];
    /* Offset: 0x100 (R/W)  Interrupt Set Pending Register */
    __IO uint32_t ISPR[1];
    uint32_t RESERVED2[31];
    /* Offset: 0x180 (R/W)  Interrupt Clear Pending Register */
    __IO uint32_t ICPR[1];
}
```

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

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

#define SCS_BASE (0xE000E000UL)

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

4.3 宏定义

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

```
/* SysTick使能控制 */
#define SysTick_Enable() (SysTick->CTRL |= 0x00000001)
#define SysTick_Disable() (SysTick->CTRL &= 0xFFFFFFF0)
```

4.4 库函数

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

4.4.1 函数NVIC_Init

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

中断源枚举类型 NVIC_IRQChannel:

枚举元素	数值	描述
NVIC_PINT0_IRQn	0	外部中断 0
NVIC_PINT1_IRQn	1	外部中断 1
NVIC_PINT2_IRQn	2	外部中断 2
NVIC_PINT3_IRQn	3	外部中断 3
NVIC_PINT4_IRQn	4	外部中断 4
NVIC_PINT5_IRQn	5	外部中断 5
NVIC_PINT6_IRQn	6	外部中断 6
NVIC_PINT7_IRQn	7	外部中断 7
NVIC_T16N0_IRQn	8	定时器中断: T16N0
NVIC_T16N1_IRQn	9	定时器中断: T16N1
NVIC_T16N2_IRQn	10	定时器中断: T16N2
NVIC_T16N3_IRQn	11	定时器中断: T16N3
NVIC_T32N0_IRQn	12	定时器中断: T32N0
NVIC_IWDT_IRQn	14	独立看门狗中断
NVIC_WWDT_IRQn	15	窗口看门狗中断
NVIC_CCM_IRQn	16	停振检测中断
NVIC_PLK_IRQn	17	PLL 失锁中断
NVIC_LVD_IRQn	18	低电压检测中断
NVIC_KINT_IRQn	19	外部按键输入中断
NVIC_RTC_IRQn	20	实时时钟中断
NVIC_ADC_IRQn	21	模数转换中断
NVIC_AES_IRQn	23	AES 加解密中断
NVIC_UART0_IRQn	24	UART0 中断
NVIC_UART1_IRQn	25	UART1 中断
NVIC_UART2_IRQn	26	UART2 中断
NVIC_UART3_IRQn	27	UART3 中断
NVIC_UART4_IRQn	28	UART4 中断
NVIC_UART5_IRQn	29	UART5 中断

NVIC_SPI0_IRQn	30	SPI0 中断
NVIC_I2C0_IRQn	31	I2C0 中断

表 4-1 NVIC_IRQChannel

中断优先级枚举类型 NVIC_IRQPriority:

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

表 4-2 NVIC_IRQPriority

4.4.2 函数SCB_SystemLPConfig

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

休眠模式枚举类型 SCB_TYPE_SCR:

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

表 4-3 SCB_TYPE_SCR

4.4.3 函数SCB_GetCpuID

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

4.4.4 函数SysTick_Init

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

初始化配置结构原型:

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

SysTick 时钟源枚举类型 SYST_TYPE_CLKS:

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

表 4-4 SYST_TYPE_CLKS

4.5 函数库应用示例

```
/* 初始化系统滴答定时器, 定时100us中断 (系统时钟为20Mhz) */
void User_SysTickInit(void)
{
    SYSTICK_InitStruType x;
    x.SysTick_ClkSource = SysTick_ClkS_Cpu; //时钟源
    x.SysTick_Value = 2000;                 //100us
    x.SysTick_ITEnable = Enable;            //中断使能
    SysTick_Init(&x);
    NVIC_Init();
    SysTick_Enable();                       //开启定时器
}
```

第5章 通用输入输出（GPIO）

5.1 功能概述

- ◆ 最多支持 45 个双向 I/O 端口
- ◆ 支持地址扩展的 I/O 端口位操作方式
- ◆ 支持 8 路外部中断输入，触发方式可配置
- ◆ 支持 8 个大电流驱动口 PA6~PA24,最大灌电流能力 40mA。

5.2 特殊说明

- ◆ 最多 45 个双向 I/O 端口（ES8P5088）
PA 端口（PA0~PA31；对 PA6 和 PA19，不同型号的芯片只支持其中一个端口）
PB 端口（PB0~PB13）
- ◆ 最多 29 个双向 I/O 端口（ES8P5086）
PA 端口（PA2~PA5, PA7~PA12, PA17~PA22, PA27~PA31）（ES8P5086FJSK）
PA 端口（PA0~PA8, PA10~PA18, PA24~PA25）（ES8P5086FJLK）
PB 端口（PB0~PB7）（ES8P5086FJSK）
PB 端口（PB0~PB7, PB11）（ES8P5086FJLK）

注意：对于不存在的引脚，请勿在程序中进行任何操作！

5.3 寄存器结构

GPIO 的寄存器定义于文件 ES8P508x.h。

```
typedef struct
{
    __IO GPIO_PAPORT_Typedef PAPORT;
    uint32_t RESERVED0[3];
    __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 RESERVED1[11];
__I GPIO_PBPORT_Typedef PBPORT;
uint32_t RESERVED2[3];
__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 RESERVED3[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 RESERVED4[139];

__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 RESERVED5[4];

__IO GPIO_IOINTFLTS_Typedef IOINTFLTS;

uint32_t RESERVED6[3];

__IO GPIO_TMRFLTSEL_Typedef TMRFLTSEL;

uint32_t RESERVED7[15];

__IO GPIO_TXPWM_Typedef TXPWM;

uint32_t RESERVED8[3];

__IO GPIO_BUZC_Typedef BUZC;

} GPIO_TypeDef;

#define APB_BASE (0x40000000UL)
#define GPIO_BASE (APB_BASE + 0x20000)
#define GPIO ((GPIO_TypeDef *) GPIO_BASE )
```

5.4 宏定义

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

```
/* PINT使能控制 */
#define PINT0_Enable() (GPIO->PINTIE.PINTIE |= 0X1)
#define PINT1_Enable() (GPIO->PINTIE.PINTIE |= 0x2)
#define PINT2_Enable() (GPIO->PINTIE.PINTIE |= 0x4)
#define PINT3_Enable() (GPIO->PINTIE.PINTIE |= 0x8)
#define PINT4_Enable() (GPIO->PINTIE.PINTIE |= 0x10)
#define PINT5_Enable() (GPIO->PINTIE.PINTIE |= 0x20)
#define PINT6_Enable() (GPIO->PINTIE.PINTIE |= 0x40)
```



```
#define PINT7_Enable() (GPIO->PINTIE.PINTIE |= 0x80)
#define PINT0_Disable() (GPIO->PINTIE.PINTIE &= ~0x01)
#define PINT1_Disable() (GPIO->PINTIE.PINTIE &= ~0x02)
#define PINT2_Disable() (GPIO->PINTIE.PINTIE &= ~0x04)
#define PINT3_Disable() (GPIO->PINTIE.PINTIE &= ~0x08)
#define PINT4_Disable() (GPIO->PINTIE.PINTIE &= ~0x10)
#define PINT5_Disable() (GPIO->PINTIE.PINTIE &= ~0x20)
#define PINT6_Disable() (GPIO->PINTIE.PINTIE &= ~0x40)
#define PINT7_Disable() (GPIO->PINTIE.PINTIE &= ~0x80)

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

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

5.5 库函数

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

5.5.1 函数GPIO_Init

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

◆ 返回值：无

初始化配置结构原型：

```
/* GPIO初始化配置结构体定义 */
typedef struct
{
    GPIO_Pin_Signal GPIO_Signal;          //引脚上的数据类型只有数字和模拟两种
    GPIO_TYPE_FUNC  GPIO_Func;           //引脚功能选择
    GPIO_TYPE_DIR   GPIO_Direction;      //方向选择
    GPIO_PUE_Input  GPIO_PUEN;           //上拉使能
    GPIO_PDE_Input  GPIO_PDEN;           //下拉使能
    GPIO_ODE_Output GPIO_OD;             //输出模式开漏使能
    GPIO_TYPE_DS    GPIO_DS;             //驱动电流控制
}GPIO_InitStruType;
```

引脚选择枚举类型 GPIO_TYPE_PIN:

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

GPIO_Pin_24	0x18	引脚 24
GPIO_Pin_25	0x19	引脚 25
GPIO_Pin_26	0x1A	引脚 26
GPIO_Pin_27	0x1B	引脚 27
GPIO_Pin_28	0x1C	引脚 28
GPIO_Pin_29	0x1D	引脚 29
GPIO_Pin_30	0x1E	引脚 30
GPIO_Pin_31	0x1F	引脚 31

表 5-1 GPIO_TYPE_PIN

引脚功能选择枚举类型 GPIO_TYPE_FUNC:

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

表 5-2 GPIO_TYPE_FUNC

方向选择枚举类型 GPIO_TYPE_DIR:

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

表 5-3 GPIO_TYPE_DIR

输出电流驱动能力选择枚举类型 GPIO_TYPE_DS:

枚举元素	数值	描述
GPIO_DS_Output_Normal	0x0	普通电流输出
GPIO_DS_Output_Strong	0x1	强电流输出

表 5-4 GPIO_TYPE_DS

信号类型选择枚举类型 GPIO_Pin_Signal:

枚举元素	数值	描述
GPIO_Pin_Signal_Digital	0x0	数字信号引脚
GPIO_Pin_Signal_Analog	0x1	模拟信号引脚

表 5-5 GPIO_Pin_Signal

弱上拉是否使能选择枚举类型 GPIO_PUE_Input:

枚举元素	数值	描述
GPIO_PUE_Input_Disable	0x0	弱上拉禁止
GPIO_PUE_Input_Enable	0x1	弱上拉使能

表 5-6 GPIO_PUE_Input

弱下拉是否使能选择枚举类型 GPIO_PUE_Input:

枚举元素	数值	描述
GPIO_PDE_Input_Disable	0x0	弱下拉禁止
GPIO_PDE_Input_Enable	0x1	弱下拉使能

表 5-7 GPIO_PUE_Input

开漏输出是否使能选择枚举类型 GPIO_ODE_Output:

枚举元素	数值	描述
GPIO_ODE_Output	0x0	弱下拉禁止
GPIO_ODE_Output	0x1	弱下拉使能

表 5-8 GPIO_ODE_Output

5.5.2 函数GPIO_Write

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

5.5.3 函数GPIO_Read

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

5.5.4 函数GPIO_ReadBit

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

5.5.5 函数GPIOA_SetBit

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

5.5.6 函数GPIOA_ResetBit

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

5.5.7 函数GPIOA_ToggleBit

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

5.5.8 函数GPIOB_SetBit

- ◆ 函数原型: void GPIOB_SetBit(GPIO_TYPE_PIN PINx)
- ◆ 功能描述: GPIOB 某引脚值 1
- ◆ 输入参数: 引脚选择, 详见表 5-1, 可以是 GPIO_Pin_0 —GPIO_Pin_13
- ◆ 返回值: 无

5.5.9 函数GPIOB_ResetBit

- ◆ 函数原型: void GPIOB_ResetBit(GPIO_TYPE_PIN PINx)
- ◆ 功能描述: GPIOB 某引脚清 0
- ◆ 输入参数: 引脚选择, 详见表 5-1, 可以是 GPIO_Pin_0 —GPIO_Pin_13
- ◆ 返回值: 无

5.5.10 函数GPIOB_ToggleBit

- ◆ 函数原型: void GPIOB_ToggleBit(GPIO_TYPE_PIN PINx)
- ◆ 功能描述: GPIOB 某引脚输出状态取反 (需先设为输出模式)
- ◆ 输入参数: 引脚选择, 详见表 5-1, 可以是 GPIO_Pin_0 —GPIO_Pin_13
- ◆ 返回值: 无

5.5.11 函数GPIOA_SetDirection

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

5.5.12 函数GPIOB_SetDirection

- ◆ 函数原型: void GPIOB_SetDirection(GPIO_TYPE_PIN PINx,
GPIO_TYPE_DIR Dir_Type)
- ◆ 功能描述: GPIOA 设置方向
- ◆ 输入参数:
 - ◇ PINx: 引脚选择, 详见表 5-1, 可以是 GPIO_Pin_0 —GPIO_Pin_13
 - ◇ Dir_Type: 引脚方向选择, 详见表 5-3
- ◆ 返回值: 无

5.5.13 函数PINT_Config

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

- ◆ 功能描述：PINT 外部中断配置
- ◆ 输入参数：
 - ◇ PINTx: 外部端口中断类型，详见表 5-9
 - ◇ SELx: 外部中断输入选择，详见表 5-10
 - ◇ TRIGx: 外部中断触发电平选择，详见表 5-11
- ◆ 返回值：无

PINT 枚举类型 PINT_TYPE:

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

表 5-9 PINT_TYPE

外部中断选择枚举类型 PINT_TYPE_SEL:

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

表 5-10 PINT_TYPE_SEL

外部中断触发电平选择枚举类型 PINT_TYPE_TRIG:

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

表 5-11 PINT_TYPE_TRIG

5.5.14 函数PINT_GetITStatus

- ◆ 函数原型: FlagStatus PINT_GetITStatus(PINT_TYPE_IT PINT_Flag)
- ◆ 功能描述: PINT 读取中断标志
- ◆ 输入参数: PINT 中断标志类型, 详见表 5-12
- ◆ 返回值: SET/RESET

PINT 中断标志枚举类型 PINT_TYPE_IT:

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

表 5-12 PINT_TYPE_IT

5.5.15 函数PINT_ClearITPendingBit

- ◆ 函数原型: void PINT_ClearITPendingBit(PINT_TYPE_IT PINT_Flag)
- ◆ 功能描述: PINT 清除中断标志
- ◆ 输入参数: PINT 中断标志类型, 详见表 5-12
- ◆ 返回值: 无

5.5.16 函数GPIO_SetFuncxRegFromPin

- ◆ 函数原型: void GPIO_SetFuncxRegFromPin(GPIO_TYPE GPIOx, GPIO_TYPE_PIN PINx, GPIO_TYPE_FUNC Func)
- ◆ 功能描述: 设置 GPIO 引脚的功能复用
- ◆ 输入参数:
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx 目的引脚, 详见表 5-1
 - ◇ Func: 功能复用编号, 详见表 5-4
- ◆ 返回值: 无

5.5.17 函数GPIO_SetSingalTypeFromPin

- ◆ 函数原型: void GPIO_SetSingalTypeFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_Pin_Signal GPIO_Signal)
- ◆ 功能描述: 设置引脚的信号类型
- ◆ 输入参数:
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚, 详见表 5-1
 - ◇ Signal: 引脚的信号类型, 详见表表 5-5
- ◆ 返回值: 无

5.5.18 函数GPIO_SetDirRegFromPin

- ◆ 函数原型: void GPIO_SetDirRegFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_TYPE_DIR Dir)
- ◆ 功能描述: 设置引脚的输入或输出方向
- ◆ 输入参数:
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚, 详见表 5-1
 - ◇ Dir: 引脚方向, 详见表 5-5
- ◆ 返回值: 无

5.5.19 函数GPIO_SetODERegFromPin

- ◆ 函数原型: void GPIO_SetODERegFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_ODE_Output ODE)
- ◆ 功能描述: 设置引脚的输出开漏方式
- ◆ 输入参数:
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚, 详见表 5-1
 - ◇ ODE: 输出开漏方式, 详见表 5-8
- ◆ 返回值: 无

5.5.20 函数GPIO_SetDSRegFromPin

- ◆ 函数原型: void GPIO_SetDSRegFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_TYPE_DS DS)

- ◆ 功能描述：设置引脚输出电流的驱动能力
- ◆ 输入参数：
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚，详见表 5-1
 - ◇ DS: 电流驱动方式，详见表 5-4
- ◆ 返回值：无

5.5.21 函数GPIO_SetPUERegFromPin

- ◆ 函数原型: void GPIO_SetPUERegFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_PUE_Input PUE)
- ◆ 功能描述：设置引脚的弱上拉方式
- ◆ 输入参数：
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚，详见表 5-1
 - ◇ PUE: 弱上拉方式，详见表 5-6
- ◆ 返回值：无

5.5.22 函数GPIO_SetPDERegFromPin

- ◆ 函数原型: void GPIO_SetPDERegFromPin(GPIO_TYPE GPIOx,GPIO_TYPE_PIN PINx, GPIO_PDE_Input PDE)
- ◆ 功能描述：PINT 清除中断标志
- ◆ 输入参数：
 - ◇ GPIOx: 可以是 GPIOA/GPIOB
 - ◇ PINx: 目的引脚，详见表 5-1
 - ◇ PDE: 弱下拉方式，详见表 5-7
- ◆ 返回值：无

5.6 函数库应用示例

```
int main(void)
{
    GPIO_InitStruType x;           //定义结构体
    SystemClockConfig();           //配置时钟
    SCU_GPIOCLK_Enable();          //使能GPIO时钟
    x.GPIO_Signal = GPIO_Pin_Signal_Digital; //信号类型
}
```

```
x.GPIO_Direction = GPIO_Dir_Out;           //端口方向
x.GPIO_Func = GPIO_Func_0;                 //端口功能
x.GPIO_OD = GPIO_ODE_Output_Enable;        //开漏
x.GPIO_DS = GPIO_DS_Output_Strong;         //电流输出方式
x.GPIO_PUEN = GPIO_PUE_Input_Enable;       //弱上拉使能
x.GPIO_PDEN = GPIO_PDE_Input_Disable;     //弱下拉禁止
GPIO_Init(GPIOA,GPIO_Pin_8,&x);            //初始化PA8
GPIOA_SetBit(GPIO_Pin_8);                 //PA8高
UserFunction1();                          //用户程序
GPIOA_ResetBit(GPIO_Pin_8);               //PA8低
UserFunction2();                          //用户程序
}
```

第6章 定时器/计数器（T16N/T32N）

6.1 功能概述

ES8P508 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

以 T16N0 为例。

- ◆ 1 个 8 位可配置预分频器，分频时钟作为 T16N_CNT 的定时/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
- ◆ 1 个 16 位可配置定时/计数寄存器 T16N_CNT
- ◆ 支持定时/计数工作模式
 - ◇ 支持 4 组 16 位计数匹配寄存器 T16N_MAT0/T16N_MAT1/T16N_MAT2/T16N_MAT3，计数匹配后支持下列操作：
 - 产生中断
 - 支持 T16N_CNT 计数寄存器三种操作：保持，清 0 或继续计数
 - 支持 T16N0OUT0/T16N0OUT1 端口四种操作：保持，清 0，置 1 或取反
- ◆ 支持输入捕捉工作模式
 - ◇ 捕捉边沿可配置
 - ◇ 捕捉次数可配置
- ◆ 支持调制工作模式
 - ◇ 通过对匹配寄存器进行配置，同时设置匹配后端口输出特性，可得到相应的 PWM 输出
 - ◇ 支持刹车控制
 - 刹车信号源可选择为 PINT0~7 中任意一个
 - 刹车信号电平极性可配置
 - 刹车信号可滤除毛刺
 - 刹车后端口输出电平可配置
- ◆ 支持计数中触发功能
 - ◇ 匹配 0、匹配 1、匹配 2、匹配 3 与 T16N_CNT 计数值匹配触发或 T16N_CNT 计数值溢出可触发 ADC 转换

6.1.2 T32N

以 T32N0 为例。

- ◆ 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 ES8P508x.h。

typedef struct

```
{  
    __IO T16N_CNT_Typedef CNT;  
    uint32_t RESERVED0;  
    __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 RESERVED1;  
    __IO T16N_IE_Typedef IE;  
    __IO T16N_IF_Typedef IF;  
    __IO T16N_TRG_Typedef TRG;  
    uint32_t RESERVED2;  
    __IO T16N_MAT0_Typedef MAT0;  
    __IO T16N_MAT1_Typedef MAT1;
```

```
__IO T16N_MAT2_Typedef MAT2;

__IO T16N_MAT3_Typedef MAT3;

} T16N_TypeDef;

typedef struct
{
    __IO T32N_CNT_Typedef CNT;
    uint32_t RESERVED0;
    __IO T32N_PRECNT_Typedef PRECNT;
    __IO T32N_PREMAT_Typedef PREMAT;
    __IO T32N_CON0_Typedef CON0;
    __IO T32N_CON1_Typedef CON1;
    __IO T32N_CON2_Typedef CON2;
    uint32_t RESERVED1;
    __IO T32N_IE_Typedef IE;
    __IO T32N_IF_Typedef IF;
    __IO T32N_TRG_Typedef TRG;
    uint32_t RESERVED2;
    __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

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

/ TIM 模块使能控制 */*

```
#define T16N0_Enable() (T16N0->CON0.EN = 1)
```

```
#define T16N1_Enable() (T16N1->CON0.EN = 1)
```

```
#define T16N2_Enable() (T16N2->CON0.EN = 1)
#define T16N3_Enable() (T16N3->CON0.EN = 1)
#define T32N0_Enable() (T32N0->CON0.EN = 1)
#define T16N0_Disable() (T16N0->CON0.EN = 0)
#define T16N1_Disable() (T16N1->CON0.EN = 0)
#define T16N2_Disable() (T16N2->CON0.EN = 0)
#define T16N3_Disable() (T16N3->CON0.EN = 0)
#define T32N0_Disable() (T32N0->CON0.EN = 0)
/* 异步写使能控制 */
#define T16N0_ASYNCWR_Enable() (T16N0->CON0.ASYWEN = 1)
#define T16N1_ASYNCWR_Enable() (T16N1->CON0.ASYWEN = 1)
#define T16N2_ASYNCWR_Enable() (T16N2->CON0.ASYWEN = 1)
#define T16N3_ASYNCWR_Enable() (T16N3->CON0.ASYWEN = 1)
#define T32N0_ASYNCWR_Enable() (T32N0->CON0.ASYNCWREN = 1)
#define T16N0_ASYNCWR_Disable() (T16N0->CON0.ASYWEN = 0)
#define T16N1_ASYNCWR_Disable() (T16N1->CON0.ASYWEN = 0)
#define T16N2_ASYNCWR_Disable() (T16N2->CON0.ASYWEN = 0)
#define T16N3_ASYNCWR_Disable() (T16N3->CON0.ASYWEN = 0)
#define T32N0_ASYNCWR_Disable() (T32N0->CON0.ASYNCWREN = 0)
/* PWM 输出使能控制 */
#define T16N0_PwmOut0_Enable() (T16N0->CON2.MOE0 = 1)
#define T16N1_PwmOut0_Enable() (T16N1->CON2.MOE0 = 1)
#define T16N2_PwmOut0_Enable() (T16N2->CON2.MOE0 = 1)
#define T16N3_PwmOut0_Enable() (T16N3->CON2.MOE0 = 1)
#define T32N0_PwmOut0_Enable() (T32N0->CON2.MOE0 = 1)
#define T16N0_PwmOut1_Enable() (T16N0->CON2.MOE1 = 1)
#define T16N1_PwmOut1_Enable() (T16N1->CON2.MOE1 = 1)
#define T16N2_PwmOut1_Enable() (T16N2->CON2.MOE1 = 1)
#define T16N3_PwmOut1_Enable() (T16N3->CON2.MOE1 = 1)
#define T32N0_PwmOut1_Enable() (T32N0->CON2.MOE1 = 1)
#define T16N0_PwmOut0_Disable() (T16N0->CON2.MOE0 = 0)
#define T16N1_PwmOut0_Disable() (T16N1->CON2.MOE0 = 0)
#define T16N2_PwmOut0_Disable() (T16N2->CON2.MOE0 = 0)
```

```
#define T16N3_PwmOut0_Disable() (T16N3->CON2.MOE0 = 0)
#define T32N0_PwmOut0_Disable() (T32N0->CON2.MOE0 = 0)
#define T16N0_PwmOut1_Disable() (T16N0->CON2.MOE1 = 0)
#define T16N1_PwmOut1_Disable() (T16N1->CON2.MOE1 = 0)
#define T16N2_PwmOut1_Disable() (T16N2->CON2.MOE1 = 0)
#define T16N3_PwmOut1_Disable() (T16N3->CON2.MOE1 = 0)
#define T32N0_PwmOut1_Disable() (T32N0->CON2.MOE1 = 0)
```

6.4 库函数

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

6.4.1 函数T16Nx_Baselnit

- ◆ 函数原型：void T16Nx_Baselnit(T16N_TypeDef* T16Nx, TIM_BaselnitStruType* TIM_BaselnitStruct)
- ◆ 功能描述：T16Nx 基本初始化
- ◆ 输入参数：
 - ◇ T16Nx：可以是 T16N0/1/2/3
 - ◇ TIM_BaselnitStruct：初始化配置结构体地址
- ◆ 返回值：无

初始化配置结构原型：

```
/* TIM初始化配置结构体定义 */
typedef struct
{
    TIM_TYPE_CLKS    TIM_ClkS;           //时钟源选择
    TYPE_FUNCEN     TIM_SYNC;           //外部时钟同步
    TIM_TYPE_EDGE    TIM_EDGE;          //外部时钟计数边沿选择
    TIM_TYPE_MODE    TIM_Mode;          //工作模式选择
}TIM_BaselnitStruType;
```

6.4.2 函数T32Nx_Baselnit

- ◆ 函数原型：T32Nx_Baselnit (T32N_TypeDef* T32Nx, TIM_BaselnitStruType* TIM_BaselnitStruct)
- ◆ 功能描述：T32Nx 基本初始化
- ◆ 输入参数：
 - ◇ T32Nx：可以是 T32N0

◇ TIM_BaselInitStruct: 初始化配置结构体地址

◆ 返回值: 无

初始化配置结构原型: 同 T16Nx

时钟源选择枚举类型 TIM_TYPE_CLKS:

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

表 6-1 TIM_TYPE_CLKS

外部时钟计数边沿选择枚举类型 TIM_TYPE_EDGE:

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

表 6-2 TIM_TYPE_EDGE

工作模式选择枚举类型 TIM_TYPE_MODE:

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

表 6-3 TIM_TYPE_MODE

6.4.3 函数T16Nx_CapInit

◆ 函数原型: void T16Nx_CapInit(T16N_TypeDef* T16Nx, TIM_CapInitStruType* TIM_CapInitStruct)

◆ 功能描述: T16Nx 捕捉功能初始化函数

◆ 输入参数:

◇ T16Nx: 可以是 T16N0/1/2/3

◇ TIM_CapInitStruct: 初始化配置结构体地址

◆ 返回值: 无

6.4.4 函数T32Nx_CapInit

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

初始化配置结构原型:

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

捕捉次数枚举类型 TIM_TYPE_CAPT:

枚举元素	数值	描述
TIM_CapTime_1	0x0	捕捉次数: 1
TIM_CapTime_2	0x1	捕捉次数: 2
TIM_CapTime_3	0x2	捕捉次数: 3
TIM_CapTime_4	0x3	捕捉次数: 4
TIM_CapTime_5	0x4	捕捉次数: 5
TIM_CapTime_6	0x5	捕捉次数: 6
TIM_CapTime_7	0x6	捕捉次数: 7
TIM_CapTime_8	0x7	捕捉次数: 8
TIM_CapTime_9	0x8	捕捉次数: 9
TIM_CapTime_10	0x9	捕捉次数: 10
TIM_CapTime_11	0XA	捕捉次数: 11
TIM_CapTime_12	0XB	捕捉次数: 12
TIM_CapTime_13	0XC	捕捉次数: 13
TIM_CapTime_14	0XD	捕捉次数: 14
TIM_CapTime_15	0xE	捕捉次数: 15
TIM_CapTime_16	0xF	捕捉次数: 16

表 6-4 TIM_TYPE_CAPT

6.4.5 函数T16Nx_MATxITConfig

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

6.4.6 函数T32Nx_MATxITConfig

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

匹配后的工作模式枚举类型 TIM_TYPE_MATCON:

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

表 6-5 TIM_TYPE_MATCON

6.4.7 函数T16Nx_MATxOutxConfig

- ◆ 函数原型:

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

6.4.8 函数T32Nx_MATxOutxConfig

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

匹配后端口的工作模式枚举类型 TIM_TYPE_MATOUT:

枚举元素	数值	描述
TIM_Out_Hold	0x0	端口: 保持
TIM_Out_Low	0x1	端口: 清 0
TIM_Out_High	0x2	端口: 置 1
TIM_Out_Switch	0x3	端口: 取反

表 6-6 TIM_TYPE_MATOUT

6.4.9 函数T16Nx_ITConfig

- ◆ 函数原型: void T16Nx_ITConfig(T16N_TypeDef* T16Nx,TIM_TYPE_IT Type,TYPE_FUNCEN NewState)
- ◆ 功能描述: T16N 中断配置
- ◆ 输入参数:
 - ◇ T16Nx: 可以是 T16N0/1/2/3

- ◇ Type: 中断类型, 详见表 6-7
- ◇ NewState: 使能/失能
- ◆ 返回值: 无

6.4.10 函数T32Nx_ITConfig

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

中断配置枚举类型 TIM_TYPE_IT:

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

表 6-7 TIM_TYPE_IT

6.4.11 函数T16Nx_PWMOutConfig

- ◆ 函数原型: void T16Nx_PWMOutConfig(T16N_TypeDef* T16Nx, T16Nx_PWMInitStruType* T16Nx_PWMInitStruct)
- ◆ 功能描述: T16N0OUT、T16N1OUT、T16N2OUT、T16N3OUT 输出配置
- ◆ 输入参数:
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ T16Nx_PWMInitStruct: 初始化结构体配置, 详见表 6-8
- ◆ 返回值: 无

初始化配置结构原型:

*/*调制功能初始化结构体定义*/*

```
typedef struct
{
    TYPE_FUNCEN T16Nx_MOE0;    //输出端口 0 使能

    TYPE_FUNCEN T16Nx_MOE1;    //输出端口 1 使能

    /* T16NxOUT0 输出极性选择位*/

    T16Nx_PWMOUT_POLAR_Type T16Nx_POL0;

    /* T16NxOUT1 输出极性选择位*/

    T16Nx_PWMOUT_POLAR_Type T16Nx_POL1;

}T16Nx_PWMInitStruType;
```

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

PWM 输出极性选择枚举类型：T16Nx_PWMOUT_POLAR_Type

枚举元素	数值	描述
POSITIVE	0x00	正极性
NEGATIVE	0x01	负极性

表 6-8 T16Nx_PWMOUT_POLAR_Type

6. 4. 12 函数T16Nx_PWMBK_Config

- ◆ 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: 可以是 T16N0/1/2/3
 - ◇ type: 配置 PWM 刹车结构体，详见表 6-8~6-11
- ◆ 返回值：无

初始化配置结构原型：

/*PWM 输出刹车控制*/

```
typedef struct
```

```
{
```

/*PWM 通道 0 刹车输出电平选择*/

T16Nx_PWMBKOUT_LEVEL T16Nx_PWMBKL0;

/*PWM 通道 1 刹车输出电平选择*/

T16Nx_PWMBKOUT_LEVEL T16Nx_PWMBKL1;

/*PWM 通道信号源选择*/

T16Nx_PWMBKP_S T16Nx_PWMBKS;

/*PWM 通道刹车信号极性选择*/

T16Nx_PWMBKP_LEVEL T16Nx_PWMBKPS;

/*PWM 刹车使能 */

TYPE_FUNCEN T16Nx_PWMBKEN;

}T16Nx_PWMBK_Type;

PWM 刹车输出电平选择枚举类型：T16Nx_PWMBKOUT_LEVEL

枚举元素	数值	描述
PWMBKOUT_Low	0	输出低电平
PWMBKOUT_High	1	输出高电平

表 6-9 T16Nx_PWMBKOUT_LEVEL

PWM 刹车信号源选择枚举类型：T16Nx_PWMBKP_S

枚举元素	数值	描述
PWMBKPS_PINT0	0	PINT0
PWMBKPS_PINT1	1	PINT1
PWMBKPS_PINT2	2	PINT2
PWMBKPS_PINT3	3	PINT3
PWMBKPS_PINT4	4	PINT4
PWMBKPS_PINT5	5	PINT5
PWMBKPS_PINT6	6	PINT6
PWMBKPS_PINT7	6	PINT7

表 6-10 T16Nx_PWMBKP_S

PWM 刹车信号极性选择枚举类型：T16Nx_PWMBKP_LEVEL

枚举元素	数值	描述
PWMBKP_High	0	高电平刹车
PWMBKP_Low	1	低电平刹车

表 6-11 T16Nx_PWMBKP_LEVEL

6.4.13 函数T16Nx_TRG_Config

- ◆ 函数原型：void T16Nx_TRG_Config(T16N_TypeDef* T16Nx,T16Nx_PWMTRG_type Type,TYPE_FUNCEN NewState)
- ◆ 功能描述：配置 T16N ADC 触发使能
- ◆ 输入参数：
 - ◇ T16Nx：可以是 T16N0/1/2/3
- ◆ 返回值：无

6.4.14 函数T16Nx_GetPWMBKF

- ◆ 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：可以是 T16N0/1/2/3
- ◆ 输出参数：PWMBKF 标志位的值。SE：发生刹车事件；RESET：未发生刹车事件
- ◆ 返回值：无

6.4.15 函数T16Nx_ResetPWMBKF

- ◆ 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 端口恢复正常输出
- ◆ 输入参数：
 - ◇ T16Nx：可以是 T16N0/1/2/3
- ◆ 返回值：无

6.4.16 函数T16Nx_SetCNT

- ◆ 函数原型：void T16Nx_SetCNT(T16N_TypeDef* T16Nx,uint16_t Value)

- ◆ 功能描述：设置计数值
- ◆ 输入参数：
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ Value: 16 位数值
- ◆ 返回值：无

6. 4. 17 函数T32Nx_SetCNT

- ◆ 函数原型：void T32Nx_SetCNT(T32N_TypeDef* T32Nx,uint32_t Value)
- ◆ 功能描述：设置计数值
- ◆ 输入参数：
 - ◇ T32Nx: 可以是 T32N0
 - ◇ Value: 32 位数值（16 位时右对齐）
- ◆ 返回值：无

6. 4. 18 函数T16Nx_SetPRECNT

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

6. 4. 19 函数T32Nx_SetPRECNT

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

6. 4. 20 函数T16Nx_SetPREMAT

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

- ◆ 功能描述：设置预分频计数匹配寄存器值
- ◆ 输入参数：
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ Value: 8 位数值
- ◆ 返回值：无

6. 4. 21 函数T32Nx_SetPREMAT

- ◆ 函数原型：void T32Nx_SetPREMAT (T32N_TypeDef* T32Nx, uint8_t Value)
- ◆ 功能描述：设置预分频计数匹配寄存器值
- ◆ 输入参数：
 - ◇ T32Nx: 可以是 T32N0
 - ◇ Value: 8 位数值
- ◆ 返回值：无

6. 4. 22 函数T16Nx_SetMATx

- ◆ 函数原型：T16Nx_SetMAT0(T16N_TypeDef* T16Nx,uint16_t Value)
T16Nx_SetMAT1(T16N_TypeDef* T16Nx,uint16_t Value)
T16Nx_SetMAT2(T16N_TypeDef* T16Nx,uint16_t Value)
T16Nx_SetMAT3(T16N_TypeDef* T16Nx,uint16_t Value)
- ◆ 功能描述：设置匹配寄存器
- ◆ 输入参数：
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ Value: 16
- ◆ 返回值：无

6. 4. 23 函数T32Nx_SetMATx

- ◆ 函数原型：T32Nx_SetMAT0(T32N_TypeDef* T32Nx,uint32_t Value)
T32Nx_SetMAT1(T32N_TypeDef* T32Nx,uint32_t Value)
T32Nx_SetMAT2(T32N_TypeDef* T32Nx,uint32_t Value)
T32Nx_SetMAT3(T32N_TypeDef* T32Nx,uint32_t Value)

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

6. 4. 24 函数T16Nx_GetMATx

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

uint16_t T16Nx_GetMAT1(T16N_TypeDef* T16Nx)

uint16_t T16Nx_GetMAT2(T16N_TypeDef* T16Nx)

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

6. 4. 25 函数T32Nx_GetMATx

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

uint32_t T32Nx_GetMAT1(T32N_TypeDef* T32Nx)

uint32_t T32Nx_GetMAT2(T32N_TypeDef* T32Nx)

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

6. 4. 26 函数T16Nx_GetCNT

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

6. 4. 27 函数T32Nx_GetCNT

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

6. 4. 28 函数T16Nx_GetPRECNT

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

6. 4. 29 函数T32Nx_GetPRECNT

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

6. 4. 30 函数T16Nx_GetFlagStatus

- ◆ 函数原型: FlagStatus T16Nx_GetFlagStatus(T16N_TypeDef* T16Nx, TIM_TYPE_IF TIM_Flag)
- ◆ 功能描述: 读取指定标志位
- ◆ 输入参数:
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ TIM_Flag: 中断标志位, 详见表 6-7

- ◆ 返回值: RESET/SET

6. 4. 31 函数T32Nx_GetFlagStatus

- ◆ 函数原型: FlagStatus T32Nx_GetFlagStatus(T32N_TypeDef* T32Nx,TIM_TYPE_IT TIM_Flag)
- ◆ 功能描述: 读取指定标志位
- ◆ 输入参数:
 - ◇ T32Nx: 可以是 T32N0
 - ◇ TIM_Flag: 中断标志位, 详见表 6-7
- ◆ 返回值: RESET/SET

6. 4. 32 函数T16Nx_GetITStatus

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

6. 4. 33 函数T32Nx_GetITStatus

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

6. 4. 34 函数T16Nx_ClearIFPendingBit

- ◆ 函数原型: T16Nx_ClearIFPendingBit(T16N_TypeDef* T16Nx,TIM_TYPE_IF TIM_Flag)
- ◆ 功能描述: 清除指定的中断标志位

- ◆ 输入参数:
 - ◇ T16Nx: 可以是 T16N0/1/2/3
 - ◇ TIM_Flag: 中断标志位, 详见表 6-7
- ◆ 返回值: 无

6.4.35 函数T32Nx_ClearIFPendingBit

- ◆ 函数原型: T32Nx_ClearIFPendingBit(T32N_TypeDef* T32Nx,TIM_TYPE_IF TIM_Flag)
- ◆ 功能描述: 清除指定的中断标志位
- ◆ 输入参数:
 - ◇ T32Nx: 可以是 T32N0
 - ◇ TIM_Flag: 中断标志位, 详见表 6-7
- ◆ 返回值: 无

6.5 函数库应用示例

```
/* T16N3定时器初始化: 5ms定时中断 */
void User_T16N3Init(void)
{
    TIM_BaselInitStruType x;
    x.TIM_ClkS = TIM_ClkS_PCLK;
    x.TIM_Mode = TIM_Mode_TCO ;
    T16Nx_BaselInit(T16N3,&x);
    T16Nx_SetPREMAT(T16N3,1);
    T16Nx_SetMAT0(T16N3,40000);
    NVIC_Init(NVIC_T16N3_IRQn,NVIC_Priority_1,ENABLE);
    T16Nx_MAT0ITConfig(T16N3,TIM_Clr_Int);
    T16Nx_ITConfig(T16N3,TIM_IT_MAT0,ENABLE);
    T16N3_Enable();
}
/* T16N3定时器初始化: 5ms定时中断 */
void T16N3_IRQHandler(void)
{
    if(T16Nx_GetFlagStatus(T16N3,TIM_IT_MAT0) != RESET)
    //判断中断类型
    {
        T16Nx_ClearIFPendingBit(T16N3,TIM_IT_MAT0);
        UserFunction();
    }
}
```

第7章 模拟/数字转换器（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 500ksps (kilo-samples per second)
- ◆ 支持最多 15 个模拟输入端
- ◆ 支持 8/10/12 位转换结果
- ◆ 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 模块的寄存器定义于文件 ES8P508x.h。芯片支持 1 个模数转换器。

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。

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

/* 自动比较功能使能 */
#define ADC_ACP_Enable() (ADC->CON0.ACP_EN = 1)
#define ADC_ACP_Disable() (ADC->CON0.ACP_EN = 0)

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

/* ADC采样软件控制 */
#define ADC_SampStart() (ADC->CON1.SMPON = 0x1)
#define ADC_SampStop() (ADC->CON1.SMPON = 0x0)

/* ADC VDD检测控制 */
#define ADC_VDD5_FLAG_Enable() (ADC->CHS.VDD5_FLAG_EN = 0x1)
#define ADC_VDD5_FLAG_Disable() (ADC->CHS.VDD5_FLAG_EN = 0x0)

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

#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 函数ADC_Init

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

- ◆ 功能描述：初始化 ADC 模块
- ◆ 输入参数：初始化配置结构体地址
- ◆ 返回值：无

初始化配置结构原型：

```
/*ADC初始化配置结构体定义*/
typedef struct

{

    ADC_TYPE_CLKS  ADC_ClkS;      //ADCCON1:bit3 ADC时钟源选择

    ADC_TYPE_CLKDIV  ADC_ClkDiv;  //ADCCON1:bit2-0 ADC时钟源预分频

    ADC_TYPE_VREFP  ADC_VrefP;    //ADCCON1:bit9-8 ADC正向参考电压选择

    ADC_TYPE_SMPS  ADC_SampS;     //ADCCON1:bit12 ADC采样模式选择

    ADC_TYPE_ST  ADC_SampClk;     //ADCCON1:bit15-14 ADC采样时间选择

    ADC_TYPE_HSEN  ADC_ConvSpeed; //ADCCON1:bit16 转换速度选择

    ADC_TYPE_CHS  ADC_ChS;        //ADCCHS:bit0-3 ADC模拟通道选择

    TYPE_FUNCEN ADC_IREF_EN;

    TYPE_FUNCEN ADC_VREF_EN;

    TYPE_FUNCEN ADC_VCMBUF_EN;

    TYPE_FUNCEN ADC_VREFN;

    TYPE_FUNCEN ADC_VRBUF_EN;

}ADC_InitStruType;
```

ADC 时钟源选择枚举类型 ADC_TYPE_CLKS:

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

表 7-1 ADC_TYPE_CLKS

ADC 时钟源预分频枚举类型 ADC_TYPE_CLKDIV:

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

表 7-2 ADC_TYPE_CLKDIV

正向参考电压选择枚举类型 ADC_TYPE_VREFP:

枚举元素	数值	描述
ADC_VrefP_Vcc	0x0	正向参考电压: 3.3V LDO 电压
ADC_VrefP_Ref	0x1	正向参考电压: 内部参考电压 Vref 2.048V , AVREFP 端口复用为普通 IO 口
ADC_VrefP_Ref2	0x2	正向参考电压: 内部参考电压 Vref 2.048V , AVREFP 端口输出 Vref
ADC_VrefP_Ext	0x3	正向参考电压: 外部参考电压

表 7-3 ADC_TYPE_VREFP

采样模式选择枚举类型 ADC_TYPE_SMPS:

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

表 7-4 ADC_TYPE_SMPS

采样时间选择枚举类型 ADC_TYPE_ST:

枚举元素	数值	描述
ADC_SampClk_2	0x0	采样时间选择: 2 个 TadClk
ADC_SampClk_4	0x1	采样时间选择: 4 个 TadClk
ADC_SampClk_8	0x2	采样时间选择: 8 个 TadClk
ADC_SampClk_16	0x3	采样时间选择: 16 个 TadClk

表 7-5 ADC_TYPE_ST

转换速度选择枚举类型 ADC_TYPE_HSEN:

枚举元素	数值	描述
------	----	----

ADC_ConvSpeed_Low	0x0	转换速度：低速
ADC_ConvSpeed_High	0x1	转换速度：高速

表 7-6 ADC_TYPE_HSEN

ADC 通道选择枚举类型 ADC_TYPE_CHS:

枚举元素	数值	描述
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

表 7-7 ADC_TYPE_CHS

7.4.2 函数ADC_Set_CH

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

7.4.3 函数ADC_GetConvValue

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

7.4.4 函数ADC_GetConvStatus

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

7.4.5 函数ADC_ACPCConfig

- ◆ 函数原型: ErrorStatus ADC_ACPCConfig(ADC_ACP_TypeDef *ADC_ACP_InitStruct)
- ◆ 功能描述: ADC 自动比较功能初始化结构体
- ◆ 输入参数: ADC_ACP_InitStruct 自动比较功能初始化结构体
- ◆ 返回值: SUCCESS / ERROR

初始化配置结构原型:

/*自动比较功能初始化结构体*/

typedef struct

{

/*自动比较功能使能位*/

ADC_TYPE_ACP_EN ACP_EN;

/*单次自动比较溢出时间（即使不想设置请设置成0）0~0x9c3*/

uint16_t ACPC_OVER_TIME;

/*单次自动比较次数（优先级高出溢出时间）*/

ADC_TYPE_ACPC_TIMES ACPC_TIMES;

/*平均值最低阈值（设置0xffff关闭）0~0xffff*/

uint16_t ACPC_MIN_TARGET;

/*平均值最高阈值（设置0x0关闭）0~0xffff*/

uint16_t ACPC_MAX_TARGET;

}ADC_ACP_TypeDef;

自动转换次数枚举类型 ADC_TYPE_ACPC_TIMES:

枚举元素	数值	描述
ADC_ACPC_TIMES_1	0x0	1 次

ADC_ACPC_TIMES_2	0x1	2 次
ADC_ACPC_TIMES_4	0x2	4 次
ADC_ACPC_TIMES_8	0x3	8 次
ADC_ACPC_TIMES_MAX	0x3	

表 7-8 ADC_TYPE_ACPC_TIMES

7.4.6 函数ADC_SampStart

- ◆ 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

7.4.7 函数ADC_SampStop

- ◆ 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

7.4.8 函数ADC_GetACPMeanValue

- ◆ 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: 采样数据

7.4.9 函数 ADC_GetACPMinValue

- ◆ 函数原型: uint16_t ADC_GetACPMinValue(void)
- ◆ 功能描述: 自动比较低阈值
- ◆ 输入参数: 无
- ◆ 返回值: RESET(完成)/SET(正在转换)

7.4.10 函数ADC_GetACPMaxValue

- ◆ 函数原型: uint16_t ADC_GetACPMaxValue(void)

- ◆ 功能描述：自动比较高阈值
- ◆ 输入参数：无
- ◆ 返回值：采样数据

7.4.11 函数ADC_GetFlagStatus

- ◆ 函数原型：FlagStatus ADC_GetFlagStatus(ADC_TYPE_IF IFName)
- ◆ 功能描述：读取 ADC 转换完成中断标志
- ◆ 输入参数：IFName ADC 中断标志位，详见表 7-9
- ◆ 返回值：RESET(完成)/SET(正在转换)

ADC IF 状态枚举类型 ADC_TYPE_IF:

枚举元素	数值	描述
ADC_IF	0x01	ADC 中断标志位
ADC_IF_ACPMIN	0x02	ADC 自动转换低阈值超出中断标志位
ADC_IF_ACPMAX	0x04	ADC 自动转换高阈值超出中断标志位
ADC_IF_ACPOVER	0x08	ADC 自动转换溢出中断标志位

表 7-9 ADC_TYPE_IF

7.4.12 函数ADC_GetITStatus

- ◆ 函数原型：ITStatus ADC_GetITStatus(ADC_TYPE_IE IENam)
- ◆ 功能描述：获取 ADC 中断状态，未使能相应中断时不会返回 SET
- ◆ 输入参数：IENam ADC 中断使能位，详见表 7-10
- ◆ 返回值：SET（中断）/RESET（无中断）

ADC IE 状态枚举类型 ADC_TYPE_IE:

枚举元素	数值	描述
ADC_IE	0x01	ADC 中断使能位
ADC_IE_ACPMIN	0x02	ADC 自动转换低阈值超出中断使能位
ADC_IE_ACPMAX	0x04	ADC 自动转换高阈值超出中断使能位
ADC_IE_ACPOVER	0x08	ADC 自动转换溢出中断使能位

表 7-10 ADC_TYPE_IE

7.4.13 函数ADC_ClearIFStatus

- ◆ 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 ADC 中断标志位, 详见表 7-9
- ◆ 返回值: SUCCESS / ERROR

7.4.14 函数ADC_Reset

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

7.5 库函数应用示例

```
/* 初始化ADC函数 */
void ADC_UserInit(void)
{
    y.ADC_ChS = ADC_CHS_AIN1;           //通道选择
    y.ADC_ClkS = ADC_ClkS_PCLK;         //时钟选择
    /*预分频选择, 请符合数据手册中ADC转化时钟源选择表相关*/
    y.ADC_ClkDiv = ADC_ClkDiv_32;
    y.ADC_VrefP = ADC_VrefP_Ref;         //正向参考电压
    y.ADC_SampS = ADC_SMPS_HARD;         //AD采样模式选择
    y.ADC_SampClk = ADC_SampClk_16;      //AD采样时间选择
    y.ADC_ConvSpeed = ADC_ConvSpeed_High; //AD转换速度
    y.ADC_IREF_EN = ENABLE;
    y.ADC_VREF_EN = ENABLE;
    y.ADC_VCMBUF_EN = ENABLE;
    y.ADC_VREFN = DISABLE;
    y.ADC_VRBUF_EN = ENABLE;
    ADC_Init(&y);
}

/* 主函数 */
int main(void)
{
    Uint16_t Temp16;
    SystemClockConfig();                 //配置时钟
    SCU_ADCCLK_Enable();                 // ADC时钟使能
    GPIO_Init();                         //GPIO初始化
    ADC_UserInit();                     //初始化ADC
    while(1)
    {
        ADC_Start();                    //ADC转换开始
        while( ==SET);                  //等待转换完成
    }
}
```

```
        ADC_ClearIFStatus();           //清除标志位
        Temp16 = ADC_GetConvValue();   //读取转换值
        UserFunction();                 //用户程序
    }
}
```


第8章 通用异步收发器（UART）

8.1 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 11 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
- ◆ 支持单线半双工异步通信模式

8.2 寄存器结构

UART 的寄存器定义于文件 ES8P508x.h。芯片支持 6 个 UART，为 UART0/1/2/3/4/5/6。

```
typedef struct
{
    __IO UART_CON_Typedef CON;
    __IO UART_BRR_Typedef BRR;
    __O UART_TBW_Typedef TBW;
    __I UART_RBR_Typedef RBR;
    __I UART_TB01_Typedef TB01;
    __I UART_TB23_Typedef TB23;
    __I UART_RB01_Typedef RB01;
    __I UART_RB23_Typedef RB23;
    __IO UART_IE_Typedef IE;
    __IO UART_IF_Typedef IF;
} UART_TypeDef;

#define APB_BASE (0x40000000UL)
#define UART0_BASE (APB_BASE + 0x06000)
#define UART1_BASE (APB_BASE + 0x06400)
#define UART2_BASE (APB_BASE + 0x06800)
#define UART3_BASE (APB_BASE + 0x06C00)
#define UART4_BASE (APB_BASE + 0x07000)
#define UART5_BASE (APB_BASE + 0x07400)
#define UART0 ((UART_TypeDef *) UART0_BASE )
#define UART1 ((UART_TypeDef *) UART1_BASE )
#define UART2 ((UART_TypeDef *) UART2_BASE )
#define UART3 ((UART_TypeDef *) UART3_BASE )
```

```
#define UART4 ((UART_TypeDef *) UART4_BASE )  
#define UART5 ((UART_TypeDef *) UART5_BASE )
```

8.3 宏定义

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

/ 发送使能控制 */*

```
#define UART0_TxEnable() (UART0->CON.TXEN = 1)  
#define UART1_TxEnable() (UART1->CON.TXEN = 1)  
#define UART2_TxEnable() (UART2->CON.TXEN = 1)  
#define UART3_TxEnable() (UART3->CON.TXEN = 1)  
#define UART4_TxEnable() (UART4->CON.TXEN = 1)  
#define UART5_TxEnable() (UART5->CON.TXEN = 1)  
#define UART0_TxDisable() (UART0->CON.TXEN = 0)  
#define UART1_TxDisable() (UART1->CON.TXEN = 0)  
#define UART2_TxDisable() (UART2->CON.TXEN = 0)  
#define UART3_TxDisable() (UART3->CON.TXEN = 0)  
#define UART4_TxDisable() (UART4->CON.TXEN = 0)  
#define UART5_TxDisable() (UART5->CON.TXEN = 0)
```

/ 接收使能控制 */*

```
#define UART0_RxEnable() (UART0->CON.RXEN = 1)  
#define UART1_RxEnable() (UART1->CON.RXEN = 1)  
#define UART2_RxEnable() (UART2->CON.RXEN = 1)  
#define UART3_RxEnable() (UART3->CON.RXEN = 1)  
#define UART4_RxEnable() (UART4->CON.RXEN = 1)  
#define UART5_RxEnable() (UART5->CON.RXEN = 1)  
#define UART0_RxDisable() (UART0->CON.RXEN = 0)  
#define UART1_RxDisable() (UART1->CON.RXEN = 0)  
#define UART2_RxDisable() (UART2->CON.RXEN = 0)  
#define UART3_RxDisable() (UART3->CON.RXEN = 0)
```

/ 发送器复位 */*

```
#define UART0_TxRst() (UART0->CON.TRST = 1)  
#define UART1_TxRst() (UART1->CON.TRST = 1)  
#define UART2_TxRst() (UART2->CON.TRST = 1)  
#define UART3_TxRst() (UART3->CON.TRST = 1)  
#define UART4_TxRst() (UART4->CON.TRST = 1)  
#define UART5_TxRst() (UART5->CON.TRST = 1)
```

/ 接收器复位 */*

```
#define UART0_RxRst() (UART0->CON.RRST = 1)  
#define UART1_RxRst() (UART1->CON.RRST = 1)
```

```
#define UART2_RxRst() (UART2->CON.RRST = 1)
#define UART3_RxRst() (UART3->CON.RRST = 1)
#define UART4_RxRst() (UART4->CON.RRST = 1)
#define UART5_RxRst() (UART5->CON.RRST = 1)
```

8.4 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。

8.4.1 函数UART_Init

- ◆ 函数原型：void UART_Init(UART_TypeDef* UARTx, UART_InitStruType* UART_InitStruct)
- ◆ 功能描述：UART 初始化
- ◆ 输入参数：
 - ◇ UARTx：可以是 UART0/1/2/3/4/5
 - ◇ UART_InitStruct：初始化配置结构体地址
- ◆ 返回值：无

初始化配置结构原型：

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

发送帧停止位选择枚举类型 UART_TYPE_TXFS:

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

表 8-1 UART_TYPE_TXFS

数据格式枚举类型 UART_TYPE_DATAMOD:

枚举元素	数值	描述
------	----	----

UART_DataMode_7	0x0	7 位数据
UART_DataMode_8	0x1	8 位数据
UART_DataMode_9	0x2	9 位数据
UART_DataMode_7Odd	0x4	7 位数据+奇校验
UART_DataMode_7Even	0x5	7 位数据+偶校验
UART_DataMode_8Odd	0x6	8 位数据+奇校验
UART_DataMode_8Even	0x7	8 位数据+偶校验

表 8-2 UART_TYPE_DATAMOD

端口极性枚举类型 UART_TYPE_RTXP:

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

表 8-3 UART_TYPE_RTXP

波特率发生器时钟选择枚举类型 UART_TYPE_BCS:

枚举元素	数值	描述
UART_Clock_1	0x1	波特率发生器时钟:PCLK
UART_Clock_2	0x2	波特率发生器时钟:PCLK/2
UART_Clock_4	0x3	波特率发生器时钟:PCLK/4
UART_Clock_8	0x4	波特率发生器时钟:PCLK/8

表 8-4 UART_TYPE_BCS

8.4.2 函数UART_ITConfig

- ◆ 函数原型: void UART_ITConfig(UART_TypeDef* UARTx, UART_TYPE_IT UART_IT, TYPE_FUNCEN NewState)
- ◆ 功能描述: UART 中断配置
- ◆ 输入参数:
 - ◇ UARTx: 可以是 UART0/1/2/3/4/5
 - ◇ UART_IT: 中断类型, 详见表 8-5
 - ◇ NewState: 使能/失能
- ◆ 返回值: 无

中断选择枚举类型 UART_TYPE_IT:

枚举元素	数值	描述
UART_IT_TB	0x0001	发送缓冲器空中断
UART_IT_RB	0x0002	接收缓冲器满中断
UART_IT_RO	0x0004	接收数据溢出中断

UART_IT_FE	0x0008	接收帧错误中断
UART_IT_PE	0x0010	接收校验错误中断
UART_IT_TBWE	0x0020	发送数据错误中断
UART_IT_TXIDLE	0x1000	发送空闲标志中断
UART_IT_RXIDLE	0x2000	接收空闲标志中断

表 8-5 UART_TYPE_IT

8.4.3 函数UART_TBIMConfig

- ◆ 函数原型: void UART_TBIMConfig(UART_TypeDef* UARTx, UART_TYPE_TRBIM Type)
- ◆ 功能描述: UART 发送缓冲器空中断模式选择
- ◆ 输入参数:
 - ◇ UARTx: 可以是 UART0/1/2/3/4/5
 - ◇ Type: 空中断模式, 详见表 9-6
- ◆ 返回值: 无

发送、接收中断模式枚举类型 UART_TYPE_TRBIM:

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

表 8-6 UART_TYPE_TRBIM

8.4.4 函数UART_RBIMConfig

- ◆ 函数原型: void UART_RBIMConfig(UART_TypeDef* UARTx, UART_TYPE_TRBIM Type)
- ◆ 功能描述: UART 接收缓冲器满中断模式选择
- ◆ 输入参数:
 - ◇ UARTx: 可以是 UART0/1/2/3/4/5
 - ◇ Type: 满中断模式, 详见表 8-6
- ◆ 返回值: 无

8.4.5 函数UART_Send

- ◆ 函数原型:
 - ◇ void UART_SendByte(UART_TypeDef* UARTx,uint8_t Temp08)
 - ◇ void UART_SendHalfWord(UART_TypeDef* UARTx,uint16_t Temp16)

◇ void UART_SendWord(UART_TypeDef* UARTx,uint32_t Temp32)

◆ 功能描述：UART 发送字节、半字、字

◆ 输入参数：

◇ UARTx：可以是 UART0/1/2/3/4/5

◇ Temp08/ Temp16/ Temp32：字节、半字、字数据

◆ 返回值：无

8.4.6 函数UART_Rec

◆ 函数原型：

◇ uint8_t UART_RecByte(UART_TypeDef* UARTx)

◇ uint16_t UART_RecHalfWord(UART_TypeDef* UARTx)

◇ uint32_t UART_RecWord(UART_TypeDef* UARTx)

◆ 功能描述：UART 接收字节、半字、字

◆ 输入参数：可以是 UART0/1/2/3/4/5

◆ 返回值：读出的字节、半字、字数据

8.4.7 函数UART_GetFlagStatus

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

◆ 功能描述：UART 获取标志位状态

◆ 输入参数：

◇ UARTx：可以是 UART0/1/2/3/4/5

◇ UART_Flag：标志位选择，详见 9-7

◆ 返回值：SET/RESET

标志位选择枚举类型 UART_TYPE_FLAG：

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

表 8-7 UART_TYPE_FLAG

8.4.8 函数UART_GetITStatus

- ◆ 函数原型: ITStatus UART_GetITStatus(UART_TypeDef* UARTx, UART_TYPE_IT UART_Flag)
- ◆ 功能描述: UART 获取中断状态,未使能相应中断时不会返回 SET
- ◆ 输入参数:
 - ◇ UARTx: 可以是 UART0/1/2/3/4/5
 - ◇ UART_Flag: 中断选择, 详见表 9-5
- ◆ 返回值: SET/RESET

8.4.9 函数UART_ClearIFPendingBit

- ◆ 函数原型: void UART_ClearIFPendingBit(UART_TypeDef* UARTx, UART_CLR_IF UART_Flag)
- ◆ 功能描述: UART 标志位清除
- ◆ 输入参数:
 - ◇ UARTx: 可以是 UART0/1/2/3/4/5
 - ◇ UART_Flag: 标志位清除选择
- ◆ 返回值: 无

标志位清除选择枚举类型 UART_CLR_IF:

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

表 8-8 UART_CLR_IF

8.4.10 函数UART_printf

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

该函数在 iDesigner 环境下支持的转义字符及格式字符(keil 环境下支持的转义字符及格式字符

同 C 库中的 printf 函数)：

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

表 8-9 UART_printf 支持的格式

8.4.11 函数 fputc

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

8.4.12 函数 static char *itoa

- ◆ 函数原型：static char *itoa(int value, char *string, int radix)
- ◆ 功能描述：将整形数据转换成字符串
- ◆ 输入参数：-radix =10 表示十进制，其他结果为 0
 - * -value 要转换的整数
 - * -buf 转换后的字符串
 - * -radix = 10
- ◆ 返回值：无

8.5 函数库应用示例

```

/* 初始化UART */
void UartInit(void)
{
    UART_InitStruType x;
    x.UART_BuadRate = 57600;
    x.UART_ClockSet = UART_Clock_1;
    x.UART_RxMode = UART_DataMode_8;
    x.UART_RxPolar = UART_Polar_Normal;
    x.UART_StopBits = UART_StopBits_1;
    x.UART_TxMode = UART_DataMode_8;
    x.UART_TxPolar = UART_Polar_Normal;
    UART_Init(UART0,&x);
    UART_RBIMConfig(UART0,UART_TBIM_Byte);
    UART_ITConfig(UART0,UART_IT_RB,Enable);
}

```

//定义结构体
 //波特率
 //时钟选择: Pclk
 //接收数据格式: 8位数据
 //接收端口极性: 正常
 //停止位: 1
 //发送数据格式: 8位数据
 //发送端口极性: 正常
 //初始化UART0
 //设置接收中断模式
 //使能接收中断


```
    NVIC_Init(NVIC_UART0_IRQn,NVIC_Priority_1,Enable); //设置优先级
}
/* printf使用示例 */
void Test_printfFunc(void)
{
    uint8_t buf = 0x55;
    printf("当前数据是(十六进制) 0x%x \r\n", buf);
    UART_printf("当前数据是(十进制) %d \r\n", buf);
}
```

第9章 IIC 串行总线

9.1 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

9.2 寄存器结构

IIC 的寄存器定义于文件 ES8P508x.h。芯片支持 1 个 IIC。

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

```

```
__IO I2C_IE_Typedef IE ;
    /* Offset: 0x00C (R/W) 中断状态寄存器 */
__IO I2C_IF_Typedef IF;
    /* Offset: 0x010 ( /W) 发送数据写入寄存器 */
__O I2C_TBW_Typedef TBW ;
    /* Offset: 0x014 (R/ ) 接收数据读取寄存器 */
__I I2C_RBR_Typedef RBR;
    /* Offset: 0x018 (R/ ) 发送缓冲寄存器 */
__I I2C_TB_Typedef TB ;
    /* Offset: 0x01C (R/ ) 接收缓冲寄存器 */
__I I2C_RB_Typedef RB ;
    /* Offset: 0x020 (R/ ) 状态寄存器 */
__I I2C_STA_Typedef STA;
} I2C_TypeDef;
#define APB_BASE (0x40000000UL)
#define I2C0_BASE (APB_BASE + 0x09000)
#define I2C0 ((I2C_TypeDef *) I2C0_BASE )
```

9.3 宏定义

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

```
/* IIC模块使能控制 */
#define I2C_Enable() (I2C0->CON.EN = 1)
#define I2C_Disable() (I2C0->CON.EN = 0)
/* IIC模块复位 */
#define I2C_Reset() (I2C0->CON.RST = 1)
/* IIC时基使能控制 */
#define I2C_TJEnable() (I2C0->CON.TJE = 1)
#define I2C_TJDisable() (I2C0->CON.TJE = 0)
/* IIC主控模式读写控制 */
#define I2C_Read() (I2C0->CON.RW = 1)
#define I2C_Write() (I2C0->CON.RW = 0)
/* IIC时钟自动下拉等待使能控制（仅从机模式支持） */
#define I2C_CSEnable() (I2C0->MOD.CSE = 1)
#define I2C_CSDisable() (I2C0->MOD.CSE = 0)
/* IIC自动发送未应答使能控制（仅从机模式支持） */
#define I2C_ANAEnable() (I2C0->MOD.ANAE = 1)
#define I2C_ANADisable() (I2C0->MOD.ANAE = 0)
/* IIC自动寻呼使能控制（仅主机模式支持） */
#define I2C_SRAEnable() (I2C0->MOD.SRAE = 1)
#define I2C_SRADisable() (I2C0->MOD.SRAE = 0)
/* IIC自动结束使能控制（仅主机模式支持） */
#define I2C_SPAEnable() (I2C->MOD.SPAE = 1)
#define I2C_SPADisable() (I2C0->MOD.SPAE = 0)
```

```
/* IIC起始位触发（仅主机模式支持） */
#define I2C_SRTrigger() (I2C0->MOD.SRT=1)
/* IIC停止位触发（仅主机模式支持） */
#define I2C_SPTrigger() (I2C0->MOD.SPT = 1)
/* IIC接收数据触发（仅主机模式支持） */
#define I2C_RDTrigger() (I2C0->MOD.RDT = 1)
/* IIC总线释放 */
#define I2C_Release() (I2C0->MOD.BLD = 1)
/* IIC发送应答设置（仅从机模式支持） */
#define I2C_TACK() (I2C0->MOD.TAS = 1)
#define I2C_TNACK() (I2C0->MOD.TAS = 0)
```

9.4 库函数

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

9.4.1 函数IIC_Init

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

初始化配置结构原型：

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

引脚开漏设置枚举类型 I2C_TYPE_PINOD：

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

表 9-1 I2C_TYPE_PINOD

工作模式选择枚举类型 I2C_TYPE_MODE:

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

表 9-2 I2C_TYPE_MODE

9.4.2 函数I2C_ITConfig

- ◆ 函数原型: void I2C_ITConfig(I2C_TYPE_IT I2C_IT, TYPE_FUNCEN NewState)
- ◆ 功能描述: I2C 中断配置
- ◆ 输入参数:
 - ◇ I2C_IT: 需要配置的中断, 详见表 9-3
 - ◇ NewState: 使能或关闭
- ◆ 返回值: 无

中断选择枚举类型 I2C_TYPE_IT:

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

表 9-3 I2C_TYPE_IT

9.4.3 函数I2C_SendAddress

- ◆ 函数原型: void I2C_SendAddress(uint8_t I2C_Address, I2C_TYPE_RWMODE Mode)
- ◆ 功能描述: I2C 发送从机地址 (主控模式时使用)
- ◆ 输入参数:
 - ◇ I2C_Address: 7 位从机地址, 左对齐 (0x00~0xFE)
 - ◇ Mode: 读或写, 详见表 9-4
- ◆ 返回值: 无

读写模式选择枚举类型 I2C_TYPE_RWMODE:

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

表 9-4 I2C_TYPE_RWMODE

9.4.4 函数 I2C_SetAddress

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

9.4.5 函数 I2C_RecModeConfig

- ◆ 函数原型: void I2C_RecModeConfig(I2C_TYPE_RECMode RecType)
- ◆ 功能描述: I2C 配置接收模式
- ◆ 输入参数:
 - ◇ RecType: 接收模式, 详见表 9-5
- ◆ 返回值: 无

接收模式选择枚举类型 I2C_TYPE_RECMode:

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

表 9-5 I2C_TYPE_RECMode

9.4.6 函数 I2C_TBIMConfig

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

- ◆ 输入参数：
 - ◇ Type: 空中断模式, 详见表 9-6
- ◆ 返回值: 无

发送、接收中断模式选择枚举类型 I2C_TYPE_TRBIM:

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

表 9-6 I2C_TYPE_TRBIM

9.4.7 函数I2C_RBIMConfig

- ◆ 函数原型: void I2C_RBIMConfig(I2C_TYPE_TRBIM Type)
- ◆ 功能描述: I2C 接收缓冲器满中断模式选择
- ◆ 输入参数:
 - ◇ Type: 满中断模式, 详见表 9-6
- ◆ 返回值: 无

9.4.8 函数I2C_AckDelay

- ◆ 函数原型: void I2C_AckDelay(I2C_TYPE_ADLY Type,TYPE_FUNCEN NewStatus)
- ◆ 功能描述: I2C 应答延时配置
- ◆ 输入参数:
 - ◇ Type: 延时时间, 详见表 9-7
 - ◇ NewStatus: 使能、失能
- ◆ 返回值: 无

应答延时选择枚举类型 I2C_TYPE_ADLY:

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

表 9-7 I2C_TYPE_ADLY

9.4.9 函数I2C_TISConfig

- ◆ 函数原型: void I2C_TISConfig(I2C_TYPE_TIS Time)
- ◆ 功能描述: I2C 数据帧传输间隔设置
- ◆ 输入参数:
 - ◇ Time: 传输间隔, 详见表 9-8
- ◆ 返回值: 无

数据帧传输间隔选择枚举类型 I2C_TYPE_TIS:

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

表 9-8 I2C_TYPE_TIS

9.4.10 函数I2C_Send

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

9.4.11 函数I2C_Rec

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

9.4.12 函数I2C_GetRWMode

- ◆ 函数原型：I2C_TYPE_RWMODE I2C_GetRWMode(void)
- ◆ 功能描述：I2C 工作读写状态读取
- ◆ 输入参数：无
- ◆ 返回值：读或写，详见表 9-4

9.4.13 函数I2C_GetTBStatus

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

9.4.14 函数I2C_GetFlagStatus

- ◆ 函数原型：FlagStatus I2C_GetFlagStatus(I2C_TYPE_FLAG I2C_Flag)
- ◆ 功能描述：I2C 获取标志位状态
- ◆ 输入参数：
 - ◇ I2C_Flag: 标志位选择，详见表 99
- ◆ 返回值：SET/RESET

标志位选择枚举类型 I2C_TYPE_FLAG:

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

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

表 9-9 I2C_TYPE_FLAG

9.4.15 函数I2C_GetITStatus

- ◆ 函数原型: FlagStatus I2C_GetITStatus(I2C_TYPE_IT I2C_Flag)
- ◆ 功能描述: I2C 获取中断状态,未使能相应中断时不会返回 SET
- ◆ 输入参数:
 - ◇ I2C_Flag: 选择需要的中断,即中断使能位,详见表 9-3
- ◆ 返回值: SET/RESET

9.4.16 函数I2C_ClearITPendingBit

- ◆ 函数原型: void I2C_ClearITPendingBit(I2C_CLR_IF I2C_IT)
- ◆ 功能描述: I2C 中断标志清除
- ◆ 输入参数:
 - ◇ I2C_IT: 选择要清除的标志位,详见表 9-10
- ◆ 返回值: 无

标志位清除选择枚举类型 I2C_CLR_IF:

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

表 9-10 I2C_CLR_IF

9.5 函数库应用示例

/* I2C1初始化程序 */

```
void User_I2CInit(void)
{
    I2C_InitStruType y;
    y.I2C_16XSamp = Disable;
    y.I2C_AutoStop = Enable;
    y.I2C_Clk = 400000;
    y.I2C_Mode = I2C_Mode_Master;
    y.I2C_SclOd = I2C_PinMode_OD;
    y.I2C_SdaOd = I2C_PinMode_OD;
    I2C_Init(&y);
    NVIC_Init(NVIC_I2C_IRQn,NVIC_Priority_1,Enable);
    I2C_ITConfig(I2C_IT_SR,Enable);
    I2C_Enable;
}
```

//定义结构
//16倍采样设置
//自动停止
//通讯频率400K
//主从设置
//端口开漏
//端口开漏
//初始化IIC1
//NVIC设置
//使能RB中断
//使能IIC1

第10章 SPI串行总线

10.1 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

10.2 寄存器结构

SPI 的寄存器定义于文件 ES8P508x.h。芯片支持 1 个 SPI。

```
typedef struct
{
    __IO SPI_CON_Typedef CON;
    uint32_t RESERVED0 ;
    __IO 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 SPI 0((SPI_TypeDef *) SPI0_BASE )
```

10.3 宏定义

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

```
/* SPI使能控制 */
```

```
#define SPI_Enable() (SPI0->CON.EN = 1)
#define SPI_Disable() (SPI0->CON.EN = 0)
/* SPI接收使能控制 */
#define SPI_RecEnable() (SPI0->CON.REN = 1)
#define SPI_RecDisable() (SPI0->CON.REN = 0)
/* SPI软件复位 */
#define SPI_Rst() (SPI0->CON.RST = 1)
/* SPI缓冲器清空 */
#define SPI_RX_CLR() (SPI0->CON.RXCLR = 1)
#define SPI_TX_CLR() (SPI0->CON.TXCLR = 1)
```

10.4 库函数

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

10.4.1 函数SPI_Init

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

初始化配置结构原型：

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

通信数据格式枚举类型 SPI_TYPE_DFS：

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

表 10-1 SPI_TYPE_DFS

通讯模式选择枚举类型 SPI_TYPE_MODE:

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

表 10-1 SPI_TYPE_MODE

10.4.2 函数SPI_ITConfig

- ◆ 函数原型: void SPI_ITConfig(SPI_TYPE_IT SPI_IE, TYPE_FUNCEN NewState)
- ◆ 功能描述: SPI 中断配置
- ◆ 输入参数:
 - ◇ SPI_IE: 中断类型, 详见表 10-3
 - ◇ NewState: 使能、失能
- ◆ 返回值: 无

中断选择枚举类型 SPI_TYPE_IT:

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

表 10-2 SPI_TYPE_IT

10.4.3 函数SPI_DataFormatConfig

- ◆ 函数原型: void SPI_DataFormatConfig(SPI_TYPE_DFS Type)
- ◆ 功能描述: SPI 数据格式配置
- ◆ 输入参数: 数据格式, 详见表 10-1
- ◆ 返回值: 无

10.4.4 函数SPI_Send

- ◆ 函数原型:
 - ◇ Void SPI_SendByte(uint8_t Temp)
 - ◇ Void SPI_SendHalfWord(uint16_t Temp)
 - ◇ Void SPI_SendWord(uint32_t Temp)

- ◆ 功能描述: SPI 发送字节、半字、字数据
- ◆ 输入参数: 字节、半字、字数据
- ◆ 返回值: 无

10.4.5 函数SPI_Rec

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

10.4.6 函数SPI_TBIMConfig

- ◆ 函数原型: void SPI_TBIMConfig(SPI_TYPE_TRBIM Type)
- ◆ 功能描述: SPI 发送缓冲器空中断模式选择
- ◆ 输入参数: 空中断模式, 详见表 10-4
- ◆ 返回值: 无

中断模式选择枚举类型 SPI_TYPE_TRBIM:

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

表 10-3 SPI_TYPE_TRBIM

10.4.7 函数SPI_RBIMConfig

- ◆ 函数原型: void SPI_RBIMConfig(SPI_TYPE_TRBIM Type)
- ◆ 功能描述: SPI 接收缓冲器满中断模式选择
- ◆ 输入参数: 满中断模式, 详见表 10-4
- ◆ 返回值: 无

10.4.8 函数SPI_GetFlagStatus

- ◆ 函数原型: FlagStatus SPI_GetFlagStatus(SPI_TYPE_FLAG Flag)

- ◆ 功能描述：SPI 读取标志位状态
- ◆ 输入参数：标志位选择，详见表 10-5
- ◆ 返回值：SET/RESET

标志位选择枚举类型 SPI_TYPE_FLAG:

枚举元素	数值	描述
SPI_Flag_TB	0x00000001	发送缓冲器空中断标志
SPI_Flag_RB	0x00000002	接收缓冲器满中断标志
SPI_Flag_TE	0x00000004	发送错误中断标志（仅从动模式支持）
SPI_Flag_RO	0x00000008	接收数据溢出中断标志
SPI_Flag_ID	0x00000010	空闲中断标志（仅主控模式支持）
SPI_Flag_NSSIF	0x00000020	片选变化中断标志（仅从动模式支持）
SPI_Flag_TBWE	0x00000040	发送数据写错误中断标志
SPI_Flag_NSS	0x00000080	片选标志（仅从动模式支持）
SPI_Flag_TBFF0	0x00000100	TB0 空标志
SPI_Flag_TBFF1	0x00000200	TB1 空标志
SPI_Flag_TBFF2	0x00000400	TB2 空标志
SPI_Flag_TBFF3	0x00000800	TB3 空标志
SPI_Flag_RBFF0	0x00001000	RB0 满标志
SPI_Flag_RBFF1	0x00002000	RB1 满标志
SPI_Flag_RBFF2	0x00004000	RB2 满标志
SPI_Flag_RBFF3	0x00008000	RB3 满标志
SPI_Flag_IDLE	0x00010000	空闲标志（仅主控模式支持）
SPI_Flag_TMS	0x00020000	帧发送间隔状态标志（仅主控模式支持）

表 10-4 SPI_TYPE_FLAG

10.4.9 函数SPI_GetITStatus

- ◆ 函数原型：ITStatus SPI_GetITStatus(SPI_TYPE_IT Flag)
- ◆ 功能描述：检查中断状态,未使能相应中断时不会返回 SET
- ◆ 输入参数：中断标志位，详见表 10-3
- ◆ 返回值：SET/RESET

10.4.10 函数SPI_ClearITPendingBit

- ◆ 函数原型：void SPI_ClearITPendingBit(SPI_CLR_IF Flag)
- ◆ 功能描述：SPI 中断标志清除
- ◆ 输入参数：标志位清除选择，详见表 10-6
- ◆ 返回值：无

标志位清除选择枚举类型 SPI_CLR_IF:

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

表 10-5 SPI_CLR_IF

10.5 函数库应用示例

```

/* SPI初始化 */
void User_SPIInit(void)
{
    SPI_InitStruType SPI_InitStruct;           //定义结构体
    SPI_InitStruct.SPI_Mode = SPI_Mode_Master; //模式设置
    SPI_InitStruct.SPI_Df = SPI_FallSendRiseRec; //设置数据格式
    SPI_InitStruct.SPI_Freq = 80000;           //波特率设置
    SPI_InitStruct.SPI_DW = 7;
    SPI_InitStruct.SPI_DelayRec = Enable;      //接收延时
    SPI_InitStruct.SPI_DelaySend = Disable;    //发送延时
    SPI_Init(&SPI_InitStruct);                //初始化SPI
    SPI_Enable();                              //使能SPI
}

```

第11章 FLASH存储器自编程（IAP）

11.1 功能概述

- ◆ 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。
- ◆ 支持页擦写保护功能，上电默认写保护使能。在 IAP 操作之前需取消目标地址页的写保护，并在完成操作后重新使能写保护，防止误擦写。
- ◆ 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。

11.2 寄存器结构

IAP 的寄存器定义于文件 ES8P508x.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;
    __IO IAP_WPROT0_Typedef WPROT0;
    __IO IAP_WPROT1_Typedef WPROT1;
    __IO IAP_WPROT2_Typedef WPROT2;
} IAP_TypeDef;

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

11.3 宏定义

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

```
/* 寄存器解锁 */
#define FlashIAP_RegUnLock() (IAP->UL.IAPUL = 0x49415055)
#define FlashIAP_RegLock() (IAP->UL.IAPUL = 0x0)

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

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

11.4 库函数

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

11.4.1 函数Flashlap_Close_WPROT

- ◆ 函数原型：ErrorStatus Flashlap_Close_WPROT(uint8_t Page)
- ◆ 功能描述：IAP 关闭写保护
- ◆ 输入参数：Page: 0-63，每 page 对应 2K 字节，64 为 INFO 区
- ◆ 返回值：成功、失败

11.4.2 函数Flashlap_Open_WPROT

- ◆ 函数原型：ErrorStatus Flashlap_Open_WPROT(uint8_t Page)
- ◆ 功能描述：IAP 开启写保护
- ◆ 输入参数：Page: 0-63，每 page 对应 2K 字节，64 为 INFO 区
- ◆ 返回值：成功、失败

11.4.3 函数Flashlap_CloseAll_WPROT

- ◆ 函数原型：ErrorStatus Flashlap_CloseAll_WPROT(void)
- ◆ 功能描述：IAP 关闭所有写保护
- ◆ 输入参数：无
- ◆ 返回值：成功、失败

11.4.4 函数Flashlap_OpenAll_WPROT

- ◆ 函数原型：ErrorStatus Flashlap_OpenAll_WPROT(void)
- ◆ 功能描述：IAP 开启所有写保护

- ◆ 输入参数：无
- ◆ 返回值：成功、失败

11.4.5 函数Flashlap_Unlock

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

11.4.6 函数Flashlap_WriteEnd

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

11.4.7 函数Flashlap_ErasePage

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

11.4.8 函数Flashlap_WriteCont

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

11.4.9 函数Flashlap_WriteWord

- ◆ 函数原型：ErrorStatus Flashlap_WriteWord(uint8_t Unit_addr, uint8_t Page_addr,

uint32_t Data32)

- ◆ 功能描述: Flash 写一个字
- ◆ 输入参数:
 - ◇ Unit_addr: 单元地址
 - ◇ Page_addr: 页地址
 - ◇ Data32: 数据
- ◆ 返回值: 成功、失败

11.4.10 函数Flash_Read

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

11.5 函数库应用示例

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

/* 判断是否写成功 */
if(Flashlap_WriteWord(0,90, Flash_WriteBuf) == ERROR)
{
    return ERROR;
}
if(Flash_Read((uint8_t *)& Flash_ReadBuf),92160,4) == ERROR)
{
    return ERROR;
}
```

第12章 硬件独立看门狗（IWDT）

12.1 功能概述

- ◆ 支持硬件使能和关闭看门狗
- ◆ IWDT 中断可唤醒深度睡眠
- ◆ IWDT 溢出时间可设定

12.2 特殊说明

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

12.3 寄存器结构

IWDT 的寄存器定义于文件 ES8P508x.h。

```
typedef struct
{
    __O IWDT_LOAD_Typedef LOAD;
    __I IWDT_VALUE_Typedef VALUE;
    __IO IWDT_CON_Typedef CON;
    __O IWDT_INTCLR_Typedef INTCLR;
    __I IWDT_RIS_Typedef RIS;
    uint32_t RESERVED0[59];
    __IO IWDT_LOCK_Typedef LOCK;
} IWDT_TypeDef;

#define APB_BASE (0x40000000UL)
#define IWDT_BASE (APB_BASE + 0x01C00)
#define IWDT ((IWDT_TypeDef *) WWDT_BASE )
```

12.4 宏定义

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

```
/* 寄存器写保护控制 */
```

```
#define IWDT_RegUnLock()    (IWDT->LOCK.Word = 0x1ACCE551)
```

```
#define IWDT_RegLock()      (IWDT->LOCK.Word = 0x0)

/* IWDT使能控制 */

#define IWDT_Enable()  {IWDT_RegUnLock();IWDT->CON.EN = 1;IWDT_RegLock();}

#define IWDT_Disable() {IWDT_RegUnLock();IWDT->CON.EN = 0;IWDT_RegLock();}

/* IWDT清除 */
#define IWDT_Clear(){IWDT_RegUnLock();IWDT->INTCLR.INTCLR =
                    0;IWDT_RegLock();}

/* IWDT中断使能控制 */

#define IWDT_ITEnable() {IWDT_RegUnLock();IWDT->CON.IE = 1;IWDT_RegLock();}

#define IWDT_ITDisable() {IWDT_RegUnLock();IWDT->CON.IE = 0;IWDT_RegLock();}

/* IWDT复位使能控制 */

#define IWDT_RstEnable() {IWDT_RegUnLock();IWDT->CON.RSTEN =
                        1;IWDT_RegLock();}

#define IWDT_RstDisable() {IWDT_RegUnLock();IWDT->CON.RSTEN =
                          0;IWDT_RegLock();}

/* IWDT计数时钟选择 */

#define IWDT_CLOCK_PCLK() {IWDT_RegUnLock();IWDT->CON.CLKS =
                          0;IWDT_RegLock();}

#define IWDT_CLOCK_WDT() {IWDT_RegUnLock();IWDT->CON.CLKS =
                          1;IWDT_RegLock();}
```

12.5 库函数

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

12.5.1 函数IWDT_Init

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

初始化配置结构原型:

```
/* IWDT初始化配置结构体定义 */
```

```
typedef struct
{
    uint32_t WDT_Tms;           //定时时间，单位ms
    TYPE_FUNCEN WDT_IE;        //中断使能
    TYPE_FUNCEN WDT_Rst;       //复位使能
    WDT_TYPE_CLKS WDT_ClockS;  //时钟选择
}WDT_InitStruType;
```

时钟选择枚举类型 WDT_TYPE_CLKS:

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

表 12-1 WDT_TYPE_CLKS

12.5.2 函数IWDT_SetReloadValue

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

12.5.3 函数IWDT_GetValue

- ◆ 函数原型: uint32_t IWDT_GetValue(void)
- ◆ 功能描述: 获取 IWDT 当前计数值
- ◆ 输入参数: 无
- ◆ 返回值: 32 位当前无符号整型数值

12.5.4 函数IWDT_GetFlagStatus

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

12.5.5 函数IWDT_GetITStatus

- ◆ 函数原型: FlagStatus IWDT_GetITStatus(void)
- ◆ 功能描述: 获取 IWDT 中断使能位状态

- ◆ 输入参数：无
- ◆ 返回值：SET/RESET

12.6 函数库应用示例

```
void WDTInit(void)
{
    IWDT_InitStruType x;           //定义结构
    IWDT_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_Clock = WDT_Clock_WDT; //时钟源选择
    IWDT_Init(&x);               //初始化WDT
    IWDT_Enable();               //使能WDT
}
```

第13章 窗口看门狗（WWDT）

13.1 功能概述

- ◆ 支持设定喂狗禁止区
- ◆ 安全可靠
- ◆ WWDT 溢出长度可设定

13.2 特殊说明

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

13.3 寄存器结构

WWDT 的寄存器定义于文件 ES8P508x.h。

```
typedef struct
{
    __O WWDT_LOAD_Typedef LOAD;
    __I WWDT_VALUE_Typedef VALUE;
    __IO WWDT_CON_Typedef CON;
    __O WWDT_INTCLR_Typedef INTCLR;
    __I WWDT_RIS_Typedef RIS;
    uint32_t RESERVED0[59];
    __IO WWDT_LOCK_Typedef LOCK;
} WWDT_TypeDef;

#define APB_BASE (0x40000000UL)
#define WWDT_BASE (APB_BASE + 0x01800)
#define WWDT ((WWDT_TypeDef *) WWDT_BASE )
```

13.4 宏定义

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

```
/* 寄存器写保护控制 */

#define WWDT_RegUnLock()    (WWDT->LOCK.Word = 0x1ACCE551)

#define WWDT_RegLock()     (WWDT->LOCK.Word = 0x0)
```

```
/* WWDT使能控制 */
#define WWDT_Enable() {WWDT_RegUnLock();WWDT->CON.EN=1;
    WWDT_RegLock();}
#define WWDT_Disable() {WWDT_RegUnLock();WWDT->CON.EN=0;
    WWDT_RegLock();}

/*W WDT清狗 */
#define WWDT_Clear() {WWDT_RegUnLock();WWDT->INTCLR.INTCLR = 0;
    WWDT_RegLock();}
/* WWDT中断使能控制 */
#define WWDT_ITEnable() {WWDT_RegUnLock();WWDT->CON.IE=1;
    WWDT_RegLock();}
#define WWDT_ITDisable() {WWDT_RegUnLock();WWDT->CON.IE=0;
    WWDT_RegLock();}

/*W WDT复位使能控制 */
#define WWDT_RstEnable() {WWDT_RegUnLock();WWDT->CON.RSTEN=1;
    WWDT_RegLock();}
#define WWDT_RstDisable() {WWDT_RegUnLock();WWDT->CON.RSTEN=0;
    WWDT_RegLock();}

/*W WDT计数时钟选择 */
#define WWDT_CLOCK_PCLK(){WWDT_RegUnLock();WWDT->CON.CLKS=0;
    WWDT_RegLock();}
#define WWDT_CLOCK_WDT(){WWDT_RegUnLock();WWDT->CON.CLKS=1;
    WWDT_RegLock();}
```

13.5 库函数

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

13.5.1 函数WWDT_Init

- ◆ 函数原型: void WWDT_Init(WWDT_InitStruType *WDT_InitStruct)
- ◆ 功能描述: WWDT 初始化
- ◆ 输入参数: 初始化配置结构体地址

◆ 返回值：无

初始化配置结构原型：

```
/*W WDT初始化配置结构体定义 */
typedef struct
{
    uint32_t WDT_Tms;           //定时时间，单位ms
    TYPE_FUNCEN WDT_IE;        //中断使能
    TYPE_FUNCEN WDT_Rst;       //复位使能
    WDT_TYPE_CLKS WDT_Clock;   //时钟选择
    WDT_TYPE_WIN WDT_Win;      //禁止喂狗窗口
}WDT_InitStruType;
```

时钟选择枚举类型 WDT_TYPE_CLKS:

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

表 13-1 WDT_TYPE_CLKS

禁止喂狗窗口枚举类型 WDT_TYPE_WIN

枚举元素	数值	描述
WDT_WIN_25	0x0	25%窗口内禁止喂狗，喂狗产生复位
WDT_WIN_50	0x1	50%窗口内禁止喂狗，喂狗产生复位
WDT_WIN_75	0x2	75%窗口内禁止喂狗，喂狗产生复位
WDT_WIN_100	0x3	不禁止喂狗，喂狗使看门狗计数器重载

表 13-2 WDT_TYPE_WIN

13.5.2 函数WWDT_SetReloadValue

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

13.5.3 函数WWDT_GetValue

- ◆ 函数原型：uint32_t WWDT_GetValue(void)
- ◆ 功能描述：获取 WWDT 当前计数值

- ◆ 输入参数：无
- ◆ 返回值：32 位当前无符号整型数值

13.5.4 函数 WWDT_GetFlagStatus

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

13.5.5 函数 Status WWDT_GetITStatus

- ◆ 函数原型：Flag Status WWDT_GetITStatus(void)
- ◆ 功能描述：获取 WWDT 中断使能位状态
- ◆ 输入参数：无
- ◆ 返回值：SET/RESET

13.6 函数库应用示例

```
void WDTInit(void)
{
    WWDT_InitStruType x;                //定义结构
    WWDT_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_Clock = WDT_Clock_WDT;        //时钟源选择
    x.WDT_Win = WDT_WIN_25;             //禁止喂狗窗口
    WWDT_Init(&x);                      //初始化WDT
    WWDT_Enable();                      //使能WDT
}
```

第14章 循环冗余校验（CRC）

14.1 功能概述

- ◆ 支持 CRC-16 和 CRC-32
- ◆ 支持 8/16/32 位宽数据
- ◆ 支持对 Flash 数据块的 CRC 校验
- ◆ 可作为通用 CRC 模块

14.2 特殊说明

CRC 模块的所有寄存器都受到了写保护。因此，所有对 CRC 模块的操作都需要先对 CRC_UL 寄存器写入 0x4352_4355，进行 CRC 解锁，操作完成后对 CRC_UL 寄存器写入其他任意值，或者 CRC 软件复位，进行 CRC 上锁。

14.3 寄存器结构

CRC 的寄存器定义于文件 ES8P508x.h。

```
typedef struct
{
    __IO CRC_UL_Typedef UL;
    __IO CRC_CON_Typedef CON;
    __IO CRC_TRIG_Typedef TRIG;
    __IO CRC_ADDR_Typedef ADDR;
    __IO CRC_SIZE_Typedef SIZE;
    __IO CRC_DI_Typedef DI;
    __I CRC_DO_Typedef DO;
    __IO CRC_STA_Typedef STA;
    __I CRC_FA_Typedef FA;
} CRC_TypeDef;

#define APB_BASE (0x40000000UL)
#define CRC_BASE (APB_BASE + 0x00C00)
#define CRC ((CRC_TypeDef *) CRC_BASE )
```

14.4 宏定义

CRC 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_crc.h 中。

例程库中没有封装 CRC 的相关宏定义。

14.5 库函数

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

14.5.1 函数CRC_getTypeValue

- ◆ 函数原型：uint32_t CRC_getTypeValue(CRC_CONValueStruType con_value)
- ◆ 功能描述：配置并返回一个 CRC_CON 配置参数
- ◆ 输入参数：con_value CRC_CON 数据配置结构体
- ◆ 返回值：CRC_CON 配置值

初始化配置结构原型：

```
/* CRC_CON 数据配置结构体初始化*/
```

```
typedef struct
```

```
{
```

```
    CRC_XOROUT xorout;    //数据输出取反选择
```

```
    CRC_REFOUT refout;    //数据输出顺序选择
```

```
    CRC_REFIN  refin;     //数据输入顺序选择
```

```
    CRC_MOD_TYPE mode;    //CRC 位宽选择
```

```
    CRC_INIT_DATA init_data_type;    //CRC 初始化数据格式选择
```

```
    CRC_HS_TYPE hs_type;    //CRC 高速模式选择
```

```
} CRC_CONValueStruType;
```

数据输出取反选择枚举类型 CRC_XOROUT：

枚举元素	数值	描述
CRC_XOROUT_NORMAL	0x0	数据输出不取反
CRC_XOROUT_ANTI	0x1	数据输出取反

表 14-1 CRC_XOROUT

数据输出顺序选择枚举类型 CRC_REFOUT

枚举元素	数值	描述
CRC_REFOUT_NORMAL	0x0	数据输出正序
CRC_REFOUT_REVERSE	0x1	数据输出倒叙

表 14-2 CRC_REFOUT

数据输入顺序选择枚举类型 CRC_REFIN

枚举元素	数值	描述
CRC_REFIN_NORMAL	0x0	数据输入正序
CRC_REFIN_REVERSE	0x1	数据输入倒叙

表 14-3 CRC_REFIN

CRC 位宽选择枚举类型 CRC_MOD_TYPE

枚举元素	数值	描述
CRC_MOD_CRC32	0x0	CRC 位宽为字节
CRC_MOD_CRC16	0x2	CRC 位宽为半字
CRC_MOD_CRC16_CCITT	0x3	CRC 位宽为字

表 14-4 CRC_MOD_TYPE

CRC 初始化数据格式选择枚举类型 CRC_INIT_DATA

枚举元素	数值	描述
CRC_INIT_DATA_ALL_0	0x0	CRC 初始化数据全为 0
CRC_INIT_DATA_ALL_1	0x2	CRC 初始化数据全为 1

表 14-5 CRC_INIT_DATA

CRC 高速模式选择枚举类型 CRC_HS_TYPE

枚举元素	数值	描述
CRC_MOD_CRC32	0x0	CRC 高速模式禁止
CRC_MOD_CRC16	0x2	CRC 高速模式使能,需CRC时钟小于24M

表 14-6 CRC_HS_TYPE

14.5.2 函数CRC_EmptyCheck

- ◆ 函数原型: uint32_t CRC_EmptyCheck(void* address, uint32_t data_len)
- ◆ 功能描述: 查空函数
- ◆ 输入参数: address: 查空区域首地址; data_len: 查空区域字节长度
- ◆ 返回值: 1: 成功, 0: 失败

14.5.3 函数CRC_FlashVerify

- ◆ 函数原型: uint32_t CRC_FlashVerify(void* address, uint32_t data_len, uint32_t type)
- ◆ 功能描述: LIASH 校验函数
- ◆ 输入参数: address: 校验区域首地址; data_len: 校验区域字节长度; type: 校验方式

配置

- ◆ 返回值: CRC 校验值

14.5.4 函数CRC_UserCal

- ◆ 函数原型: uint32_t CRC_UserCal(void* address, uint32_t data_len, uint32_t type)
- ◆ 功能描述: 用户数据校验函数
- ◆ 输入参数: address: 用户校验数据首地址; data_len: 校验区域字节长度; type: 校验方式配置
- ◆ 返回值: CRC 校验值

14.5.5 函数CRC_CheckReset

- ◆ 函数原型: uint32_t CRC_CheckReset(void)
- ◆ 功能描述: CRC 复位查询函数
- ◆ 输入参数: 无
- ◆ 返回值: 复位标志, 1: 有复位标志, 0: 无复位标志

14.6 函数库应用示例

```
/*CRC演示例程*/
int test_0(void)
{
    uint8_t in_data[6] = {0x11,0x22,0x33,0x44,0x55,0x66};
    uint32_t out_data =0;
    CRC_CONValueStruType x;           //定义结构体
    uint32_t type;
    x.xorout = CRC_XOROUT_NORMAL;     //输出数据不取反
    x.refout = CRC_REFOUT_REVERSE;    //数据输出倒序
    x.refin = CRC_REFIN_REVERSE;      //数据输入倒序
    x.mode = CRC_MOD_CRC16;           //CRC位宽为半字
    x.init_data_type = CRC_INIT_DATA_ALL_0; //CRC初始化数据全为0
    x.hs_type = CRC_HS_TYPE_DISABLE;  //CRC高速模式禁止
    type = CRC_getTypeValue(x);
    out_data = CRC_UserCal(in_data,6,type);
    if(out_data == 0x6d0b)
        return 1;
    else
        return 0;
}
```

第15章 循环冗余校验（CRC）

15.1 功能概述

AES（Advanced Encryption Standard）是 1997 年美国 ANSI 向全球发起征集加密算法作为数据加密标准，最后 Rijindael 算法入选。

15.2 寄存器结构

AES 的寄存器定义于文件 ES8P508x.h。

```
typedef struct
{
    __IO AES_DATA0_Typedef DATA0;
    __IO AES_DATA1_Typedef DATA1;
    __IO AES_DATA2_Typedef DATA2;
    __IO AES_DATA3_Typedef DATA3;
    __IO AES_KEY0_Typedef KEY0;
    __IO AES_KEY1_Typedef KEY1;
    __IO AES_KEY2_Typedef KEY2;
    __IO AES_KEY3_Typedef KEY3;
    __IO AES_CON_Typedef CON;
} AES_TypeDef;

#define APB_BASE (0x40000000UL)
#define AES_BASE (APB_BASE + 0x0A000)
#define AES ((AES_TypeDef *) AES_BASE )
```

15.3 宏定义

AES 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_aes.h 中。

```
/*加解密使能，开始加解密*/
```

```
#define AES_Enable() (AES->CON.GO_DONE = 1)
```

```
/*加密模块关闭*/
```

```
#define AES_Disable() (AES->CON.GO_DONE = 0)
```

15.4 库函数

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

15.4.1 函数AES_Init

- ◆ 函数原型: void AES_Init(AES_InitStruType * AES_InitStruct)
- ◆ 功能描述: AES 功能初始化函数
- ◆ 输入参数: AES_InitStruct 初始化结构体
- ◆ 返回值:

初始化配置结构原型:

```
/* 加解密初始化结构体*/
```

```
typedef struct {
```

```
AES_TYPE_MODE MODE;    //加密或者解密
```

```
}AES_InitStruType;
```

加密解密模式选择枚举类型 AES_TYPE_MODE:

枚举元素	数值	描述
AES_MODE_DECRYPT	0	解密
AES_MODE_ENCRYPT	1	加密

表 15-1 AES_TYPE_MODE

15.4.2 函数AES_WriteKey

- ◆ 函数原型: void AES_WriteKey(uint32_t *AES_KEY)
- ◆ 功能描述: AES 写入密钥函数
- ◆ 输入参数: AES_KEY 待写入密钥
- ◆ 返回值: 无

15.4.3 函数AES_ReadKey

- ◆ 函数原型: void AES_ReadKey (uint32_t *AES_KEY)
- ◆ 功能描述: AES 读出密钥函数
- ◆ 输入参数: AES_KEY 读出密钥存放位置
- ◆ 返回值: 无

15.4.4 函数AES_WriteData

- ◆ 函数原型: void AES_WriteData(uint32_t *AES_DATA)

- ◆ 功能描述: AES 写入数据函数
- ◆ 输入参数: AES_DATA 待写入数据
- ◆ 返回值: 无

15.4.5 函数AES_ReadData

- ◆ 函数原型: void AES_ReadData(uint32_t * AES_DATA)
- ◆ 功能描述: AES 读出数据函数
- ◆ 输入参数: AES_DATA 读出数据存放位置
- ◆ 返回值: 无

15.4.6 函数AES_ITConfig

- ◆ 函数原型: void AES_ITConfig(AES_TYPE_IT AES_IE, TYPE_FUNCEN NewState)
- ◆ 功能描述: AES 中断使能
- ◆ 输入参数: AES_IE 详见表 15-2, NewState Enable/Disable
- ◆ 返回值: 无

AES 中断使能位 AES_TYPE_IT:

枚举元素	数值	描述
AES_IT_IT	0x40	AES 中断使能位

表 15-2 AES_TYPE_IT

15.4.7 函数AES_GetFlagStatus

- ◆ 函数原型: FlagStatus AES_GetFlagStatus(AES_TYPE_IF IFName)
- ◆ 功能描述: AES 获得特定中断标志函数
- ◆ 输入参数: IFName AES_IF_IF 加解密完成中断, 详见表 15-3
- ◆ 返回值: SET/RESET

AES 中断标志位 AES_TYPE_IF:

枚举元素	数值	描述
AES_IF_IF	0x80	AES 中断标志

表 15-3 AES_TYPE_IF

15.4.8 函数AES_ClearITPendingBit

- ◆ 函数原型: void AES_ClearITPendingBit(void)

- ◆ 功能描述: AES 清除特定中断标志函数
- ◆ 输入参数: 无
- ◆ 返回值: 无

15.4.9 函数AES_GetDoneStatus

- ◆ 函数原型: AES_TYPE_DONE AES_GetDoneStatus(void)
- ◆ 功能描述: AES 获得是否加/解密完成
- ◆ 输入参数: AES_DATA 读出数据存放位置
- ◆ 返回值: AES_DONE_NO : 加密未完成

AES_DONE_YES : 未加密或加密已完成 详见表 15-4

加密解密控制位判断枚举类型 AES_TYPE_DONE:

枚举元素	数值	描述
AES_DONE_YES	0	未加密或加密已完成
AES_DONE_NO	1	加密未完成

表 15-4 AES_TYPE_DONE

15.4.10 函数AES_Reset

- ◆ 函数原型: void AES_Reset(void)
- ◆ 功能描述: AES 复位
- ◆ 输入参数: 无
- ◆ 返回值: 无

15.5 函数库应用示例

```

/*AES演示例程*/
AES_InitStruType x;
/*加密 */
x.MODE = AES_MODE_ENCRYPT;
AES_Init(&x);

AES_WriteKey(key);
AES_WriteData(data);
AES_Enable();
while( AES_GetDoneStatus() == AES_DONE_NO);
AES_ReadData(res0);
    
```

```
/* 解密 */  
x.MODE = AES_MODE_DECRYPT;  
AES_Init(&x);  
AES_WriteKey(key);  
AES_WriteData(res0);  
AES_Enable();  
while( AES_GetDoneStatus() == AES_DONE_NO);  
AES_ReadData(res1);
```

第16章 实时时钟（RTC）

16.1 功能概述

- ◆ 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：工作电压为 3.6V 时模块工作电流典型值为 1.5 μA （最大值为 3 μA ）

16.2 寄存器结构

RTC 的寄存器定义于文件 ES8P508x.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

RTC 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_rtc.h` 中。

例程库中没有封装 RTC 的相关宏定义。

16.4 库函数

RTC 库函数定义于 `lib_rtc.c` 中，声明于 `lib_rtc.h` 中。

16.4.1 函数 `RTC_Init`

- ◆ 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，详见表 16-1；

HOURx: 12/24 小时制选择，详见表 16-2

- ◆ 返回值：无

RTC 时钟源选择枚举类型 `RTC_TYPE_CLKS`:

枚举元素	数值	描述
<code>RTC_LOSC</code>	0x0	外部 32KHz 时钟，精准计数
<code>RTC_LRC</code>	0x1	内部 LRC 时钟，非精准计数
<code>RTC_PCLK_256</code>	0x2	PCLK/256 RTC 用作普通计数器
<code>RTC_PCLK</code>	0x3	PCLK RTC 用作普通计数器

表 16-1 `RTC_TYPE_CLKS`

RTC 时钟 12/24 小时制选择枚举类型 `RTC_TYPE_TIME`:

枚举元素	数值	描述
<code>RTC_HOUR12</code>	0x0	12 小时制
<code>RTC_HOUR24</code>	0x1	24 小时制

表 16-2 `RTC_TYPE_TIME`

16.4.2 函数 `RTC_StartRead`

- ◆ 函数原型：void `RTC_StartRead(void)`
- ◆ 功能描述：启动实时时钟读流程
- ◆ 输入参数：无
- ◆ 返回值：无

16.4.3 函数RTC_ReadHourmode

- ◆ 函数原型: uint32_t RTC_ReadHourmode(void)
- ◆ 功能描述: 读小时模式
- ◆ 输入参数: 无
- ◆ 返回值: 当前小时模式

16.4.4 函数RTC_ReadSecond

- ◆ 函数原型: uint32_t RTC_ReadSecond (void)
- ◆ 功能描述: 读取秒
- ◆ 输入参数: 无
- ◆ 返回值: 当前的秒

16.4.5 函数RTC_ReadMinute

- ◆ 函数原型: uint32_t RTC_ReadMinute(void)
- ◆ 功能描述: 读取分
- ◆ 输入参数: 无
- ◆ 返回值: 当前的分

16.4.6 函数RTC_ReadHour

- ◆ 函数原型: uint32_t RTC_ReadHour(uint32_t *meridiem)
- ◆ 功能描述: 读取小时
- ◆ 输入参数: meridiem: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 当前的小时

16.4.7 函数RTC_ReadDay

- ◆ 函数原型: uint32_t RTC_ReadDay(void)
- ◆ 功能描述: 读取日
- ◆ 输入参数: 无
- ◆ 返回值: 当前的日

16.4.8 函数RTC_ReadMonth

- ◆ 函数原型: uint32_t RTC_ReadMonth (void)
- ◆ 功能描述: 读取月
- ◆ 输入参数: 无
- ◆ 返回值: 当前的月

16.4.9 函数RTC_ReadYear

- ◆ 函数原型: uint32_t RTC_ReadYear (void)
- ◆ 功能描述: 读取年
- ◆ 输入参数: 无
- ◆ 返回值: 当前的年

16.4.10 函数RTC_ReadWeek

- ◆ 函数原型: uint32_t RTC_ReadWeek(void)
- ◆ 功能描述: 读取星期
- ◆ 输入参数: 无
- ◆ 返回值: 当前的星期

16.4.11 函数RTC_Start Write

- ◆ 函数原型: void RTC_Start Write (void)
- ◆ 功能描述: 启动实时时钟写流程
- ◆ 输入参数: 无
- ◆ 返回值: 无

16.4.12 函数RTC_WriteSecond

- ◆ 函数原型: ErrorStatus RTC_WriteSecond(uint32_t second)
- ◆ 功能描述: 修改秒
- ◆ 输入参数: second: 秒
- ◆ 返回值: 无

16. 4. 13 函数RTC_WriteMinute

- ◆ 函数原型: `ErrorStatus RTC_Write Minute (uint32_t minute)`
- ◆ 功能描述: 修改分
- ◆ 输入参数: `minute`: 分
- ◆ 返回值: 无

16. 4. 14 函数RTC_WriteHour

- ◆ 函数原型: `ErrorStatus RTC_WriteHour(uint32_t hour, uint32_t meridiem)`
- ◆ 功能描述: 修改时
- ◆ 输入参数: `hour`: 时; `meridiem`: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 无

16. 4. 15 函数RTC_WriteDay

- ◆ 函数原型: `ErrorStatus RTC_WriteDay(uint32_t day)`
- ◆ 功能描述: 修改天
- ◆ 输入参数: `day`: 天
- ◆ 返回值: 无

16. 4. 16 函数RTC_Write Month

- ◆ 函数原型: `ErrorStatus RTC_Write Month (uint32_t month)`
- ◆ 功能描述: 修改月
- ◆ 输入参数: `month`: 月
- ◆ 返回值: 无

16. 4. 17 函数RTC_WriteYear

- ◆ 函数原型: `ErrorStatus RTC_WriteYear(uint32_t year)`
- ◆ 功能描述: 修改年
- ◆ 输入参数: `year`: 年
- ◆ 返回值: 无

16.4.18 函数RTC_WriteWeek

- ◆ 函数原型: `ErrorStatus RTC_WriteWeek(uint32_t week)`
- ◆ 功能描述: 修改星期
- ◆ 输入参数: `week`: 星期 (0-6)
- ◆ 返回值: 无

16.4.19 函数RTC_ReadWeekAlarmMinute

- ◆ 函数原型: `uint32_t RTC_ReadWeekAlarmMinute(void)`
- ◆ 功能描述: 读取分
- ◆ 输入参数: 无
- ◆ 返回值: 当前的分

16.4.20 函数RTC_ReadWeekAlarmHour

- ◆ 函数原型: `uint32_t RTC_ReadWeekAlarmHour(uint32_t *meridiem)`
- ◆ 功能描述: 读取小时
- ◆ 输入参数: `meridiem`: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 当前的小时

16.4.21 函数RTC_ReadWeekAlarmWeek

- ◆ 函数原型: `uint32_t RTC_ReadWeekAlarmWeek(void)`
- ◆ 功能描述: 读取星期
- ◆ 输入参数: 无
- ◆ 返回值: 当前的星期

16.4.22 函数RTC_ReadDayAlarmMinute

- ◆ 函数原型: `uint32_t RTC_ReadDayAlarmMinute(void)`
- ◆ 功能描述: 读取分
- ◆ 输入参数: 无
- ◆ 返回值: 当前的分

16. 4. 23 函数RTC_ReadDayAlarmHour

- ◆ 函数原型: uint32_t RTC_ReadDayAlarmHour(uint32_t *meridiem)
- ◆ 功能描述: 读取小时
- ◆ 输入参数: meridiem: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 当前的小时

16. 4. 24 函数RTC_WriteWeekAlarmMinute

- ◆ 函数原型: ErrorStatus RTC_WriteWeekAlarmMinute(uint32_t minute)
- ◆ 功能描述: 修改分钟
- ◆ 输入参数: minute: 分钟
- ◆ 返回值: 无

16. 4. 25 函数RTC_WriteWeekAlarmHour

- ◆ 函数原型: ErrorStatus RTC_WriteWeekAlarmHour(uint32_t hour, uint32_t meridiem)
- ◆ 功能描述: 修改分钟
- ◆ 输入参数: hour: 小时; meridiem: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 无

16. 4. 26 函数RTC_WriteWeekAlarmWeek

- ◆ 函数原型: ErrorStatus RTC_WriteWeekAlarmWeek(uint32_t week)
- ◆ 功能描述: 修改星期
- ◆ 输入参数: week: 星期 (0-6)
- ◆ 返回值: 无

16. 4. 27 函数RTC_WriteDayAlarmMinute

- ◆ 函数原型: ErrorStatus RTC_WriteDayAlarmMinute(uint32_t minute)
- ◆ 功能描述: 修改分钟
- ◆ 输入参数: minute: 分钟
- ◆ 返回值: 无

16.4.28 函数RTC_WriteDayAlarmHour

- ◆ 函数原型: `ErrorStatus RTC_WriteDayAlarmHour(uint32_t hour, uint32_t meridiem)`
- ◆ 功能描述: 修改分钟
- ◆ 输入参数: hour: 小时; meridiem: 0/AM, 1/PM 仅在 12 小时模式有效
- ◆ 返回值: 无

16.4.29 函数RTC_InterruptEnable

- ◆ 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, 详见表 16-3
- ◆ 返回值: 无

RTC 中断源选择枚举类型 `RTC_Interrupt_Source`:

枚举元素	数值	描述
<code>RTC_Interrupt_Source_Second</code>	0	秒中断
<code>RTC_Interrupt_Source_Minute</code>	1	分中断
<code>RTC_Interrupt_Source_Hour</code>	2	小时中断
<code>RTC_Interrupt_Source_Day</code>	3	天中断
<code>RTC_Interrupt_Source_Month</code>	4	月中断
<code>RTC_Interrupt_Source_DayALE</code>	5	日闹铃中断
<code>RTC_Interrupt_Source_WeekALE</code>	6	周闹铃中断

表 16-3 `RTC_Interrupt_Source`

16.4.30 函数RTC_InterruptDisable

- ◆ 函数原型: `void RTC_InterruptDisable(RTC_Interrupt_Source src)`
- ◆ 功能描述: 禁止实时时钟的某些中断
- ◆ 输入参数: src: 实时时钟的中断源, 详见表 16-3
- ◆ 返回值: 无

16.4.31 函数RTC_GetITStatus

- ◆ 函数原型: `ITStatus RTC_GetITStatus(RTC_Interrupt_Source src)`
- ◆ 功能描述: 获取实时时钟的某些中断状态
- ◆ 输入参数: src: 实时时钟的中断源, 详见表 16-3
- ◆ 返回值: 中断状态

16.4.32 函数RTC_GetFlagStatus

- ◆ 函数原型: FlagStatus RTC_GetFlagStatus(RTC_Interrupt_Source src)
- ◆ 功能描述: 读取实时时钟的某些中断标志
- ◆ 输入参数: src: 实时时钟的中断源, 详见表 16-3
- ◆ 返回值: 中断标志

16.4.33 函数RTC_ClearAllITFlag

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

16.5 函数库应用示例

```
/*实时时钟读写演示*/
uint32_t wa_min,

/*读取上电默认值 */
wa_min = RTC_ReadWeekAlarmMinute();

/* 写入新值*/
RTC_WriteWeekAlarmMinute(25);
wa_min = RTC_ReadWeekAlarmMinute();
```

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：

1. 计算 BRR 寄存器值

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

若 $BRR > 2047$ ，则 $BRR = 2047$ 。

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

2. 计算真实波特率

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

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。

3. 计算误差

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

下表为 F_{pclk} 为 20MHz 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	9	125000	9%	4	125000	9%	1	156250	36%
57600	20	59523	3%	9	62500	9%	4	62500	9%
38400	31	39062	2%	15	39062	2%	7	39062	2%
19200	64	19230	0%	31	19531	2%	15	19531	2%
14400	85	14534	1%	42	14534	1%	20	14880	3%
9600	129	9615	0%	64	9615	0%	31	9765	2%
7200	172	7225	0%	85	7267	1%	42	7267	1%
4800	259	4807	0%	129	4807	0%	64	4807	0%
3600	346	3602	0%	172	3612	0%	85	3633	1%
2400	519	2403	0%	259	2403	0%	129	2403	0%
1800	693	1801	0%	346	1801	0%	172	1806	0%
1200	1040	1200	0%	519	1201	0%	259	1201	0%
600	2047	610	0%	1040	600	0%	519	600	0%
300	2047	610	103%	2047	305	2%	1040	300	0%

150	2047	610	307%	2047	305	103%	2047	152	1%
-----	------	-----	------	------	-----	------	------	-----	----

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：

1. 计算 TJP 寄存器值

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

若 $TJP > 255$ ，则 $TJP = 255$ 。

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 IIC_16XSamp 参数有关，若 $\text{IIC_16XSamp} = \text{Disable}$ ， $n = 16$ ；若 $\text{IIC_16XSamp} = \text{Enable}$ ， $n = 24$ 。

2. 计算真实波特率

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

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

3. 计算误差

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

下表为 F_{pclk} 为 20MHz 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	2	416666	4%	400000	1	416666	4%
350000	2	416666	19%	350000	1	416666	19%
300000	3	312500	4%	300000	1	416666	39%
250000	4	250000	0%	250000	2	277777	11%
200000	5	208333	4%	200000	3	208333	4%
150000	7	156250	4%	150000	4	166666	11%
100000	11	104166	4%	100000	7	104166	4%
80000	14	83333	4%	80000	9	83333	4%
60000	19	62500	4%	60000	12	64102	7%
50000	24	50000	0%	50000	15	52083	4%
40000	30	40322	1%	40000	19	41666	4%
20000	61	20161	1%	20000	40	20325	2%
10000	124	10000	0%	10000	82	10040	1%
5000	249	5000	0%	5000	165	5020	1%
1000	255	4882	388%	1000	255	3255	226%

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：

1. 计算 CKS 寄存器值

$$CKS = \text{INT} \left(\frac{F_{\text{pclk}}}{\text{Dbaud} \times 2} \right)$$

若 $CKS > 255$ ，则 $CKS = 255$ 。

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

2. 计算真实波特率

$$\text{Rbaud} = \frac{F_{\text{pclk}}}{CKS \times 2}$$

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

3. 计算误差

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

下表为 F_{pclk} 为 20MHz 时，一些典型波特率的误差。

Fpclk=16 000 000			
Dbaud	CKS	Rbaud	误差
16000000	0	20000000	25%
15000000	0	20000000	33%
10000000	1	10000000	0%
8000000	1	10000000	25%
6000000	1	10000000	67%
4000000	2	5000000	25%
2000000	5	2000000	0%
1000000	10	1000000	0%
800000	12	833333	4%
600000	16	625000	4%
400000	25	400000	0%
200000	50	200000	0%
100000	100	100000	0%
80000	125	80000	0%
60000	166	60240	0%
40000	250	40000	0%
20000	255	39215	96%

表 16-3 SPI 波特率误差