UM3506 SDK 参考手册

版本：V1.1

广芯微电子（广州）股份有限公司

http://www.unicmicro.com/

条款协议

本文档的所有部分，其著作产权归广芯微电子（广州）股份有限公司（以下简称广芯微电子）所有，未经广芯微电子授权许可，任何个人及组织不得复制、转载、仿制本文档的全部或部分组件。本文档没有任何形式的担保、立场表达或其他暗示，若有任何因本文档或其中提及的产品所有资讯所引起的直接或间接损失，广芯微电子及所属员工恕不为其担保任何责任。除此以外，本文档所提到的产品规格及资讯仅供参考，内容亦会随时更新，恕不另行通知。

1. 本文档中所记载的关于电路、软件和其他相关信息仅用于说明半导体产品的操作和应用实例。用户如在设备设计中应用本文档中的电路、软件和相关信息，请自行负责。对于用户或第三方因使用上述电路、软件或信息而遭受的任何损失，广芯微电子不承担任何责任。
2. 在准备本文档所记载的信息的过程中，广芯微电子已尽量做到合理注意，但是，广芯微电子并不保证这些信息都是准确无误的。用户因本文档中所记载的信息的错误或遗漏而遭受的任何损失，广芯微电子不承担任何责任。
3. 对于因使用本文档中的广芯微电子产品或技术信息而造成的侵权行为或因此而侵犯第三方的专利、版权或其他知识产权的行为，广芯微电子不承担任何责任。本文档所记载的内容不应视为对广芯微电子或其他人所有的专利、版权或其他知识产权作出任何明示、默示或其它方式的许可及授权。
4. 使用本文档中记载的广芯微电子产品时，应在广芯微电子指定的范围内，特别是在最大额定值、电源工作电压范围、热辐射特性、安装条件以及其他产品特性的范围内使用。对于在上述指定范围之外使用广芯微电子产品而产生的故障或损失，广芯微电子不承担任何责任。
5. 虽然广芯微电子一直致力于提高广芯微电子产品的质量和可靠性，但是，半导体产品有其自身的具体特性，如一定的故障发生率以及在某些使用条件下会发生故障等。此外，广芯微电子产品均未进行防辐射设计。所以请采取安全保护措施，以避免当广芯微电子产品在发生故障而造成火灾时导致人身事故、伤害或损害的事故。例如进行软硬件安全设计（包括但不限于冗余设计、防火控制以及故障预防等）、适当的老化处理或其他适当的措施等。

版本修订

版本	日期	描述
V1.0	2019.08.31	首次正式版本
V1.1	2022.07.22	更新工程、API 描述等

目录

1 概述

本文档旨在指导用户如何使用广芯微的 UM3506 SDK 的 API，通过一些简单的示例展示如何使用 UM3506 微处理器的通用 IO 接口、PWM、ADC 等功能。

UM3506 SoC 芯片支持以下功能：

- 增强型 PD3.0 PD/Type-C 控制器
  - TCPM/TCPC 内部架构，允许多端口扩展
  - 集成了 1 native TCPC
    - CC 检测和控制逻辑
    - PD BMC PHY
    - Part of PD protocol
  - 全功能 PD3.0
    - PPS，260bytes 长包，
    - FRS 等
  - SRC/SNK/DRP 能力
  - 支持 QC4.0+/Apple 快充协议
- 高性能 RISC-V RV32IMC 32bit MCU core
- 片内 Flash/SRAM
- 增强型外设接口
  - SPI，2* I2C，2 * UART，GPIO
- 6 路增强型 PWM
- 12bit SAR ADC，高达 16 通道
- LSCSA OCP，OVP
- 集成 TL431 for SRC 产品
- 支持 3.3~24V VBUS 供电
- 支持低功耗模式
- FreeRTOS 操作系统支持
- PD+ 的应用
  - 充电器
  - 多端口移动电源
  - PD+无线充

2 架构

UM3506 采用 RISC-V 32 位指令集内核。
内核遵循的标准：
- RISC-V 特权架构文档版本为:“特权架构文档版本 1.10”(riscv-privileged-v1.10.pdf)
- RISC-V 指令集文档版本为:“指令集文档版本 2.2”(riscv-spec-v2.2.pdf)

用户可以在 RISC-V 基金会的网站上下载其完整原文(https://riscv.org/specifications/)。

UM3506 内核特性如下：
- 哈弗架构（Harvard Architecture：程序指令和数据分开）
- 机器特权模式（Machine Mode）
- 小端字节序（Little-Endian Order）
- RISC-V 指令集支持 RV32IMC：
  - RV32 架构：32 位地址空间，通用寄存器宽度 32 位
  - I：支持 32 个通用整数寄存器，47 个整形指令
  - M：支持 8 个整数乘法/除法指令
  - C：支持 27 个编码长度为 16 位的压缩指令，提高代码密度
- 8 根外部中断线
- 支持 JTAG 调试接口，2 个硬件断点
- 内嵌 3 个 64 位性能计数器
  - 实时时钟计数器
  - 时钟周期计数器
  - 指令完成计数器

2.1 内核 CSR 寄存器

RISC-V 的架构中定义了一些控制和状态寄存器（Control and Status Register - CSR）用于配置或者记录一些运行的状态。CSR 寄存器是处理器核内部的寄存器，使用其专有的 12 位地址编码空间。

以下符号用于指定 CSR 寄存器内的位域属性：
- RO - 只读
- QRO - 只读（写尝试被忽略）
- RW - 读/写
- RZ - 读取为零
- W1S - 写 1 设置

核心实现了 CSR 访问的以下规则:
1. 试图访问不存在的 CSR 会引发非法指令异常；
2. 试图编写只读 CSR 也会引发非法指令异常；
3. 如果读/写寄存器包含一些只读位，则忽略对只读位的写。

CSR 定义如**错误!未找到引用源。**所示。

表 2-1 内核 CSR 寄存器

CSR	地址	读写属性	名称	全称
标准**CSR**	0xC00	只读	cycle	周期计数器的低32位（Lower 32 bits of User-mode Cycle counter）

	0xC01	只读	time	定时器寄存器的低32位（Lower 32 bits of User-mode timer register）
	0xC02	只读	instret	完成指令计数器的低32位（Lower 32 bits of User-mode Instructions-retired counter）
	0xC80	只读	cycleh	周期计数器的高32位（Upper 32 bits of User-mode Cycle counter）
	0xC81	只读	timeh	定时器寄存器的高32位（Upper 32 bits of User-mode timer register）
	0xC82	只读	instreth	完成指令计数器的高32位（Upper 32 bits of User-mode Instructions-retired counter）
	0xF11	只读	mvendorid	商业供应商编号寄存器（Machine Vendor ID Register）
	0xF12	只读	marchid	架构编号寄存器（Machine Architecture ID Register）
	0xF13	只读	mimpid	硬件实现编号寄存器（Machine Implementation ID Register）
	0xF14	只读	mhartid	Hart编号寄存器（Hart ID Register）
	0x300	读写	mstatus	状态寄存器（Machine Status Register）
	0x301	只读	misa	指令集架构寄存器（Machine ISA Register）
	0x304	读写	mie	局部中断屏蔽控制寄存器（Machine Interrupt Enable Register）
	0x305	读写	mtvec	异常入口基地址寄存器
	0x340	读写	mscratch	暂存寄存器（Machine Scratch Register）
	0x341	读写	mepc	异常PC寄存器（Machine Exception Program Counter）
	0x342	读写	mcause	异常原因寄存器（Machine Cause Register）
	0x343	读写	mtval	异常值寄存器（Machine Trap Value Register）
	0x344	读写	mip	中断等待寄存器（Machine Interrupt Pending Register）
	0xB00	读写	mcycle	周期计数器的低32位（Lower 32 bits of Machine-mode Cycle counter）
	0xB80	读写	mcycleh	周期计数器的高32位（Upper 32 bits of Machine-mode Cycle counter）
	0xB02	读写	minstret	完成指令计数器的低32位（Lower 32 bits of Machine-mode Instructions-retired counter）
	0xB82	读写	minstreth	完成指令计数器的高32位（Upper 32 bits of Machine-mode Instructions-retired counter）
非标准 CSR	0x7E0	读写	MCOUNTEN	计数器使能
	0xBF0	只读	IPIC_CISV	当前运行中的中断向量寄存器
	0xBF1	读写	IPIC_CICSR	当前运行中的中断状态控制寄存器
	0xBF2	读写	IPIC_IPR	中断等待寄存器
	0xBF3	只读	IPIC_ISVR	在运行的中断寄存器（含等待中的）
	0xBF4	读写	IPIC_EOI	中断停止
	0xBF5	读写	IPIC_SOI	中断开始
	0xBF6	读写	IPIC_IDX	中断向量索引

	0xBF7	读写	IPIC_ICSR	中断状态控制寄存器
内存映射 CSR	0x00490000	读写	TIMER_CTRL	定时器控制寄存器（Machine-mode timer control register）
	0x00490004	读写	TIMER_DIV	定时器分频寄存器（Machine-mode timer divider register）
	0x00490008	读写	mtime	定时器计数寄存器的低32位（Lower 32 bits of Machine-mode timer register）
	0x0049000C	读写	mtimeh	定时器计数寄存器的高32位（Upper 32 bits of Machine-mode timer register）
	0x00490010	读写	mtimecmp	定时器比较寄存器的低32位（Lower 32 bits of Machine-mode timer compare register）
	0x00490014	读写	mtimecmph	定时器比较寄存器的高32位（Upper 32 bits of Machine-mode timer compare register）

标准 CSR 详情请参考 RISC-V 特权架构文档（riscv-privileged-v1.10.pdf）。

2.1.1 MCOUNTEN [0x7E0]

MCOUNTEN CSR 允许通过软件禁用应用程序不需要的计数器。如果禁用了 CYCLE[H]和 INSTRET[H] CSR，则这两个 CSR 不可读。MCOUNTEN 寄存器的结构如**错误!未找到引用源。**所示。

表 2-2 MCOUNTEN 寄存器结构

位	域	属性	描述
0	CY	RW	Enable cycle counter
1	RSV	RZ	Reserved
2	IR	RW	Enable retired instructions counter
31..3	RSV	RZ	Reserved

2.1.2 IPIC 寄存器 [0xBF0..0xBF7]

0xBF0..0xBF7 寄存器是**可编程中断控制器**（Integrated Programmable Interrupt Controller - IPIC）定义的非标准 CSR，详情请看 中断机制 章节。

2.1.3 TIMER_CTRL [0x00490000]

TIMER_CTRL CSR 允许软件启用内核**内部定时器**功能并选择时钟源。TIMER_CTRL 寄存器的结构如**错误!未找到引用源。**所示。

表 2-3 TIMER_CTRL 寄存器结构

位	域	属性	描述
0	ENABLE	RW	Timer enable
1	CLKSRC	RW	Timer clock source: 0 - internal core clock (default) 1 - external real-time clock

31..2		RZ	Reserved, read as zero

2.1.4 TIMER_DIV [0x00490004]

TIMER_DIV CSR 允许软件设定内核**内部定时器**分频器。TIMER_DIV 寄存器的结构如表 20 所示。

表 2-4 TIMER_DIV 寄存器结构

位	域	属性	描述
9..0	DIV	RW	Timer divider: timer tick occurs every DIV+1 clock ticks
31..10		RZ	Reserved, read as zero

2.2 异常机制

内核在执行程序指令的过程中突然遇到了异常的事情而中止执行当前的程序，转而去处理该异常，其要点如下：

- 异常是由内核内部事件或程序执行中的事件引起的，譬如本身硬件故障、程序故障，或者执行特殊的系统服务指令而引起的，简而言之是一种内因。
- 异常不可以被屏蔽，异常发生后，内核一定会进入异常服务处理程序。

表 2-5 MCAUSE 寄存器中的 EXCEPTION CODE

异常编号	异常和中断类型	描述
0	指令地址非对齐（Instruction address misaligned）	指令PC地址非对齐。 注意：该异常类型在UM3506中不可能发生。
1	指令访问错误（Instruction access fault）	取指令访存错误。
2	非法指令（Illegal instruction）	非法指令。
3	断点（Breakpoint）	RISC-V架构定义了EBREAK指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往用于调试器（Debugger）使用，譬如设置断点
4	读存储器地址非对齐（Load address misaligned）	Load指令访存地址非对齐。 注意：UM3506内核不支持地址非对齐的数据存储器读写操作，因此当访问地址非对齐时会产生此异常。
5	读存储器访问错误（Load access fault）	Load指令访存错误。
6	写存储器和AMO地址非对齐（Store/AMO address misaligned）	Store或者AMO指令访存地址非对齐。注意：UM3506内核不支持地址非对齐的数据存储器读写操作，因此当访问地址非对齐时会产生此异常。
7	写存储器和AMO访问错误（Store/AMO access fault）	Store或者AMO指令访存错误。
8	用户模式环境调用（Environment call from U-mode）	User Mode下执行ecall指令。 RISC-V架构定义了ecall指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往供软件使用，强行进入异常模式。
11	机器模式环境调用（Environment call from M-mode）	Machine Mode下执行ecall指令。 RISC-V架构定义了ecall指令，当处理器执行到该指令时，会发生异常进入异常服务程序。该指令往往供软件使用，强行进入异常模式。

2.3 中断机制

中断（Interrupt）机制，即处理器内核在顺序执行程序指令流的过程中突然被别的请求打断而中止执行当前的程序，转而去处理别的事情，待其处理完了别的事情，然后重新回到之前程序中断的点继续执行之前的程序指令流。

中断的若干基本知识要点如下：

- 打断处理器当前执行的请求便称之为中断请求（Interrupt Request，IRQ），中断请求的来源便称之为中断源，中断源通常来自于内核外部（称之为外部中断源），也可以来自于内核内部（成为内部中断源）。
- 处理器转而去处理的中断请求便称之为中断服务程序（Interrupt Service Routine，ISR）。
- 中断处理是一种正常的机制，而非一种错误情形。处理器收到中断请求之后，需要保存当前程序的现场，简称为“保存现场”。等到处理完中断服务程序后，处理器需要恢复之前的现场，从而继续执行之前被打断的程序，简称为“恢复现场”。

在 UM3506 中，中断源包含如下几个：

- 内部异常中断，优先级最高
- 外部中断，优先级次高
- 内部定时器中断，优先级次低
- 内部软件中断，优先级最低

全局中断使能通过 mstatus 寄存器控制。

2.3.1 内部异常中断

内部异常中断是执行**异常处理机制**（详情请看 2.2 异常机制），具有最高优先级。
在 SDK 的中由 trap_handler 函数负责打印输出。

2.3.2 外部中断

UM3506 核心集成了**可编程中断控制器**（Integrated Programmable Interrupt Controller，IPIC），UM3506 外部中断产生的中断请求均由**可编程中断控制器**处理，**IPIC** 具有低延迟的中断请求响应。可以使用 **IPIC 控制状态寄存器**（）配置 IPIC。

中断向量是指 IPIC 响应外部中断而产生的外部中断数。

UM3506 支持最多 8 个中断向量[0…7]和 8 个中断线[0…7]，每条线都被静态地映射到对应的向量，映射表如下：

表 2-6 中断向量映射表

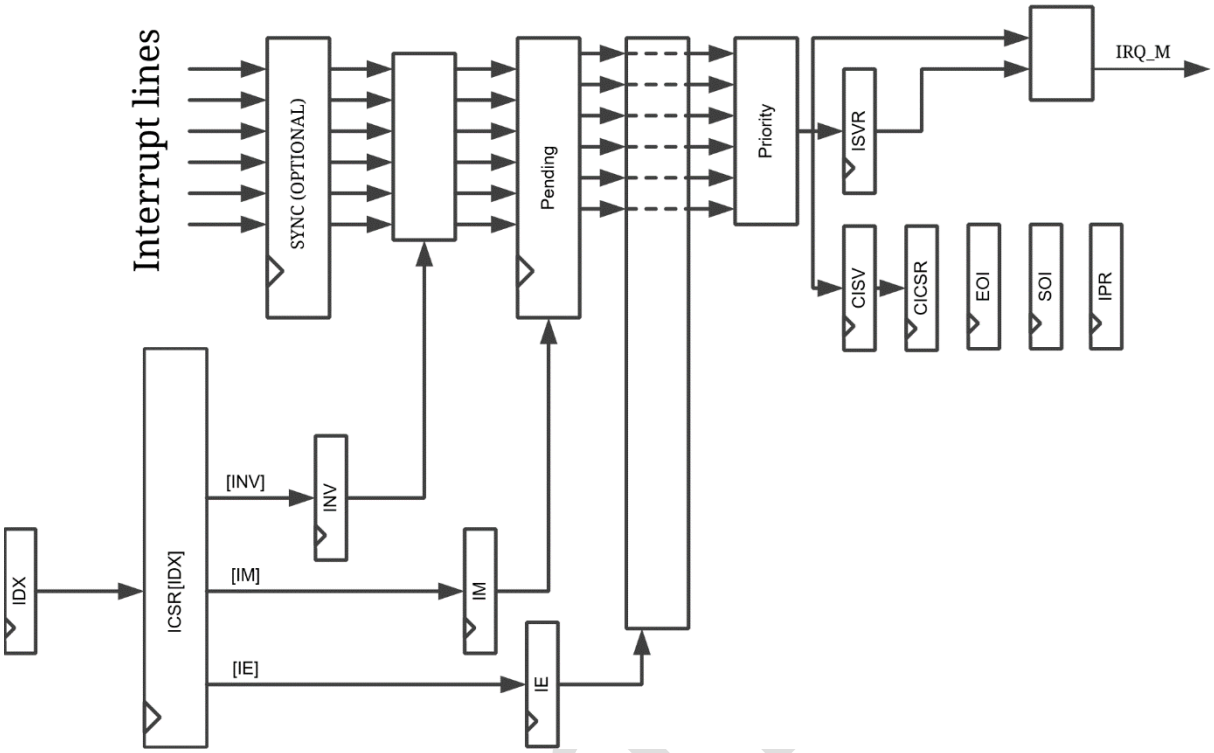
中断向量	中断线
IRQ[7]	GPIO 中断
IRQ[6]	SPI 中断
IRQ[5]	UART1 中断
IRQ[4]	UART0 中断
IRQ[3]	I2C1 中断
IRQ[2]	I2C0 中断
IRQ[1]	DMA 中断
IRQ[0]	PD 中断

IPIC 中断向量具有优先级。默认情况下，最小的中断向量值具有最高优先级。应用程序也可以在系统初始化时重新设定中断向量的优先级，详情请参考**错误!未找到引用源。**。

IPIC 支持嵌套中断。一次只能服务一个中断。

2.3.2.1 IPIC 框图及说明

图 2-1 IPIC 框图



根据每个向量的 IM（断模式）、INV（line inversion）值，选择 IP（Interrupt Pending）位激活的四个条件之一：高电平、低电平、上升沿、下降沿。在所有具有 IP 和 IE（中断启用）位活动的向量中，编号最小的向量具有最高的优先级（默认配置）。软件负责编写 SOI 和 EOI 寄存器，从而分别通知 IPIC 中断处理的开始和结束。

2.3.2.2 IPIC 编程模型

2.3.2.2.1 寄存器映射

以下符号用于指定 IPIC 寄存器内的位域属性：

- RO - 只读
- RW - 读/写
- R/W1S -读/写 1 设置
- R/W1C -读/写 1 清除

IPIC 控制状态寄存器文件访问权限由当前的特权模式定义。所有寄存器只能从机器模式访问。

机器模式下的 IPIC 寄存器相对于 CSR 空间中给定的 IPIC 基址偏移 0xBF0 进行映射，如**错误!未找到引用源。**所示。

表 2-7 IPIC 寄存器映射

偏移	名称	描述
0x00	IPIC_CISV	Current Interrupt Vector in Service
0x01	IPIC_CICSR	Current Interrupt Control Status Register
0x02	IPIC_IPR	Interrupt Pending Register

0x03	IPIC_ISVR	Interrupts in Service Register
0x04	IPIC_EOI	End Of Interrupt
0x05	IPIC_SOI	Start of Interrupt
0x06	IPIC_IDX	Index Register
0x07	IPIC_ICSR	Interrupt Control Status Register

2.3.2.2.2 IPIC 寄存器详细描述

2.3.2.2.2.1 IPIC_CISV: 当前服务中的中断向量

IPIC_CISV 寄存器结构如**错误!未找到引用源。**所示。

表 2-8 IPIC_CISV 寄存器结构

位	属性	描述
4..0	QRO	Number of the interrupt vector currently in service
31..5	RZ	Reserved

IPIC_CISV 寄存器包含当前正在服务的中断向量的数量（也是 IPIC_ISVR 中分配的最低位的数量）。当没有中断服务时，这个寄存器包含空中断向量的数量（0x10）。

2.3.2.2.2.2 IPIC_CICSR: 当前服务中的中断控制状态寄存器

IPIC_CICSR 寄存器的结构如**错误!未找到引用源。**所示。

表 2-9 IPIC_CICSR 寄存器的结构

位	域	属性	描述
0	IP	R/W1C	Interrupt pending:
			0 - no interrupt
			1 - Interrupt pending
1	IE	RW	Interrupt Enable Bit:
			0 - Interrupt disabled
			1 - Interrupt enabled
31..2	Reserved	RZ	Reserved

控制状态寄存器，用于当前正在服务的中断向量。

如果当前没有中断服务时，读寄存器将返回 0。

2.3.2.2.2.3 IPIC_IPR: 中断挂起寄存器

IPIC_IPR 寄存器结构如**错误!未找到引用源。**所示。

表 2-10 IPIC_IPR 寄存器结构

位	属性	描述

0	RW1C	Interrupt vector 0 pending status (1- pending)
1	RW1C	Interrupt vector 1 pending status (1- pending)
…		
15	RW1C	Interrupt vector 15 pending status (1- pending)
31..16	RZ	Reserved

所有挂起中断的聚合状态。对于挂起的中断，相应的位被设置为 1。

2.3.2.2.2.4 IPIC_ISVR: 服务状态寄存器

IPIC_ISVR 寄存器的结构如**错误!未找到引用源。**所示。

表 2-11 IPIC_ISVR 寄存器结构

位	属性	描述
0	QRO	Interrupt vector 0 processing status (1- in service)
1	QRO	Interrupt vector 1 processing status (1- in service)
…		
15	QRO	Interrupt vector 15 processing status (1- in service)
31..16	RZ	Reserved

包含中断向量的聚合状态，这些中断向量当前处于服务状态。换句话说，包括嵌套中断在内的所有已经开始处理但尚未完成的向量。

当相应的位被设置(1)时——这个中断向量处于服务状态。当对应的位在 0 时——中断向量不工作。

2.3.2.2.2.5 IPIC_EOI: 结束中断服务寄存器

IPIC_EOI 寄存器的结构如**错误!未找到引用源。**所示。

表 2-12 IPIC_EOI 寄存器结构

位	属性	描述
31..0	RZW	End-of-interrupt (any value can be written)

将任何值写入 EOI 寄存器将终止当前正在服务的中断。寄存器值被更新以反映状态变化:

- IPIC_CISV 被设置为它的前一个值，如果一些中断在当前中断之前是活动的，否则设置为 0x10。
- 如果在当前中断之前某个中断是活动的，则将 IPIC_CICSR 设置为其以前的值，否则设置为 0。
- IPIC_ISVR：清除当前中断对应的位。

2.3.2.2.2.6 IPIC_SOI: 开始中断服务寄存器

IPIC_EOI 寄存器的结构如**错误!未找到引用源。**所示。

表 2-13 IPIC_SOI 寄存器结构

位	属性	描述
31..0	RZW	Start-of-interrupt (any value can be written)

如果下列条件为真，则将任何值写入 SOI 将激活中断启动：

- IE 至少有一个挂起的中断，ISR 为零（没有中断服务）。
- IE 至少有一个挂起的中断，这个中断的优先级比当前服务中的中断高。

寄存器值被更新以反映状态变化：

- IPIC_CISV 被设置为最高优先级的挂起中断号。
- IPIC_CICSR 被设置为反映最高优先级挂起中断的值。
- IPIC_IPR：清除与最高优先级挂起中断对应的位。
- IPIC_ISVR：设置一个与最高优先级挂起中断相对应的位。

2.3.2.2.2.7 IPIC_IDX: 中断向量的索引寄存器

IPIC_IDX 寄存器的结构如**错误!未找到引用源。**所示。

表 2-14 IPIC_IDX 寄存器结构

位	属性	描述
3..0	RW	Interrupt vector index to access through IPIC_ICSR
31..4	RZ	Reserved

IPIC_IDX 寄存器中的值定义了通过 IPIC_ICSR 寄存器访问的中断向量的索引。

2.3.2.2.2.8 IPIC_ICSR: 中断控制状态寄存器

IPIC_ICSR 寄存器的结构如**错误!未找到引用源。**所示。

表 2-15 IPIC_ICSR 寄存器

位	域	属性	描述
0	IP	RW1C	Interrupt pending:
			0 - no interrupt
			1 - Interrupt pending
1	IE	RW	Interrupt Enable Bit:
			0 - Interrupt disabled
			1 - Interrupt enabled
2	IM	RW	Interrupt Mode:
			0 - Level interrupt
			1 - Edge interrupt
3	INV	RW	Line Inversion:
			0 - no inversion
			1 - line inversion
4	IS	RW	In Service
7..5	Reserved	RZ	
9..8	PRV	QRO	Privilege mode: hardwired to 11 (machine mode)
11..10	Reserved	RZ	

15..12	LN	QRO	External IRQ Line Number assigned to this interrupt vector. This value is always equal to IPIC_IDX, because of the static line to vector mapping.
31..16	Reserved	RZ	

这是中断向量的控制状态寄存器，由索引寄存器(IPIC_IDX)定义。

2.3.3 内部定时器中断

内核 **内部定时器** 支持通过 mtime 和 mtimecmp 寄存器比较产生定时器中断。

只有当全局中断被启用并且在 mie 寄存器中设置了 MTIE 位时，才会产生中断。

当 mtime 寄存器（64 位，含 mtimeh）的值大于或等于 mtimecmp 寄存器（64 位，含 mtimecmph）中的值时，就会产生定时器中断。中断将一直被触发，直到中断关闭或通过编写 mtimecmp 寄存器清除它为止。

SDK 中需要打开 SYS_DRIVER_M_TIMER_ENABLED 定义，系统层需要设置自己的中断服务程序。

2.3.4 内部软件中断

RISC-V 架构定义了一种软件中断，可以通过软件设置 mie 寄存器的 MSIE 位来触发。中断产生后，需要软件清除 mie 寄存器的 MSIE 位来关闭中断。

SDK 中，系统层需要设置自己的中断服务程序。

2.4 定时器

UM3506 提供了两种类型的硬件定时器：

- CPU 内核 **内部定时器**
- CPU 外部 **自定义定时器**

内部定时器 只有一个，是 RISC-V 架构定义的定时器，通过内部定时器中断触发中断，详情请参考 2.3.3。

自定义定时器 有 3 个，目前 SDK 只用了一个，详情请参考 4.1.13。

2.5 系统存储

2.5.1 数据访问宽度和对齐

UM3506 内核支持以下内存访问宽度:

- 32 位字可以用于指令和数据存储
- 16 位半字仅用于数据内存
- 8 位字节仅用于数据内存

数据内存是一个连续的字节集合，按升序编号，范围为 0x60000000-0xFFFFFFFF(32 位地址)。

指令内存可以看作是基本 32 位指令集(RV32I)的 32 位字的连续集合，或者紧凑指令集(RV32C)的 16 位半字的连续集合。指令内存必须相应地对齐到 4 字节边界或 2 字节边界。内存中的字节编号从 0 开始。

2.5.2 内存映射

UM3506 内核支持程序指令与数据分开的体系结构，能有效降低指令与数据访问的时延。

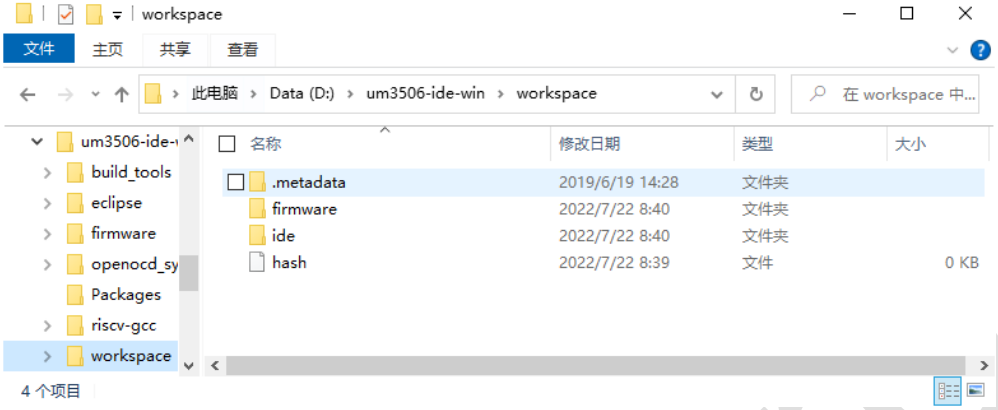
错误!未找到引用源。显示了 UM3506 的内存映射。

图 2-2 内存映射表

详细定义请看“UM3506 用户手册 V1.3.pdf”。

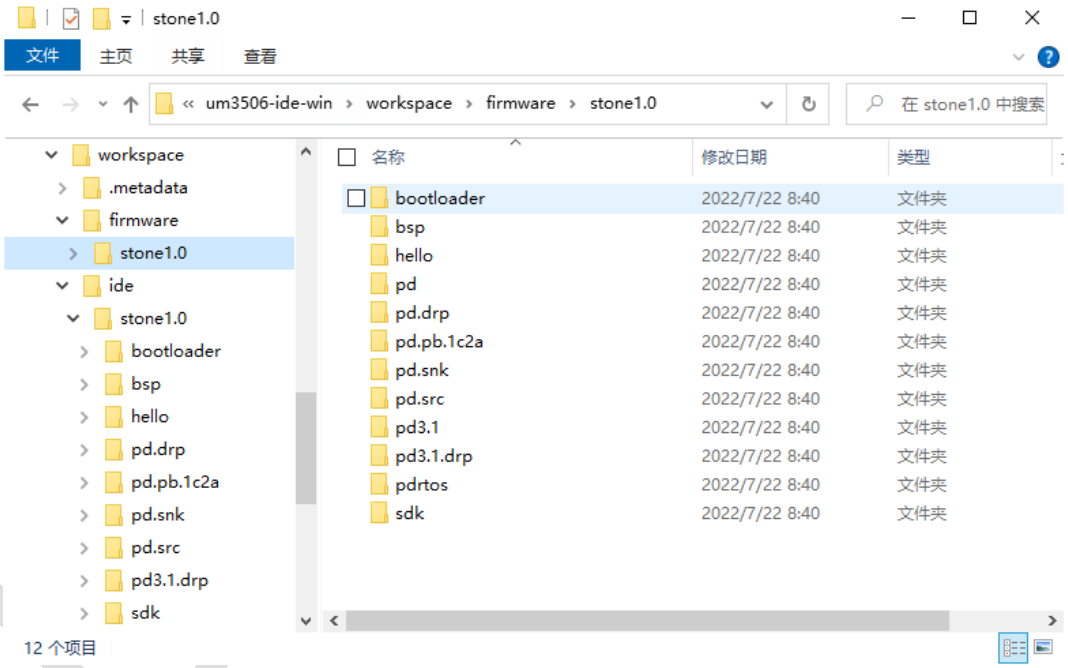
3 代码结构

在 workspace 目录下有两个主要目录，如下图：



其中，firmware 目录是存放源代码的，ide 目录是存放 Eclipse 工程文件的。

目录展开后，如下：



注意

每个应用程序工程必须包含 platform_config.h 文件，文件内容请参考.\firmware\stone1.0\bootloader 或 sdk 等其他应用工程。

3.1 bsp 库

bsp（board support package）是一个 **静态库** 工程，所有应用程序都应该要链接 bsp 库。

代码由 arch、drivers、freertos 以及 libc 四个部分组成。其中：

- arch：MCU 架构相关，由 RISC-V 的 CSR 寄存器访问接口、LD 脚本与启动汇编组成。

- drivers：MCU 驱动，ADC、DMA、FLASH、GPIO、I2C、SPI…等。
- freertos：FreeRTOS 是一款适用于微控制器的开源操作系统，可通过宏 SYS_FREERTOS_ENABLED 设置 0 或 1 来关闭或使能 FreeRTOS。
- libc：自定义 C 库，如果需要更多 C 库支持，可以使用目前 **工具链** 支持的 Newlib C 库。

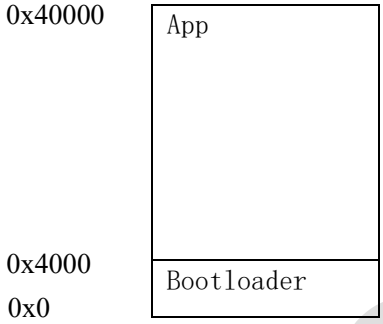
3.2 bootloader

bootloader 工程是引导装载程序。需要考虑在线升级的可能需要引导装载程序。

目前支持功能：

- 无操作系统
- 启动后检测 UART0 是否有 x 字符输入，如果有则启动 xmodem 协议升级功能
- 如果没有 x 字符输入，500 毫秒后自动启动 0x4000 地址后的应用程序空间

目前 FLASH 空间划分：



代码包含：

- main.c：入口实现，链接到.\firmware\stone1.0\bootloader\main.c
- platform_config.h：
  配置文件，链接到.\firmware\stone1.0\bootloader\platform_config.h
- bsp：bsp 子目录，链接到.\firmware\stone1.0\bsp 目录

工程编译后的固件**（也就是烧录文件）**：
.\ide\stone1.0\bootloader\Debug\bootloader.bin

3.3 hello

UM3506 的 helloworld 工程。是一个最简单应用的例子：

代码包含：

- main.c：helloworld 的实现，链接到.\firmware\stone1.0\hello\main.c
- platform_config.h：配置文件，链接到.\firmware\stone1.0\hello\platform_config.h
- bsp：bsp 子目录，链接到.\firmware\stone1.0\bsp 目录

工程编译后的固件：
.\ide\stone1.0\hello\Debug\hello.bin

3.4 sdk

sdk 工程是 UM3506 SoC 的软件开发工具包的非 PD 应用例子。
目前支持功能：

- FreeRTOS
- CLI 命令行接口
- 所有外设测试接口
- 可扩展的系统配置保存
- 低功耗模式

代码包含：

- main.c：sdk 的 demo 入口实现，链接到.\firmware\stone1.0\sdk\main.c
- platform_config.h：配置文件，链接到.\firmware\stone1.0\sdk\platform_config.h
- cli：cli 子目录，链接到.\firmware\stone1.0\sdk\cli 目录
  - include\cli_cmd_base.h
  - include\cli_engine.h
  - cli_cmd_base.c
  - cli_engine.c
- system：system 子目录，链接到.\firmware\stone1.0\sdk\system 目录
  - include\sys_config.h
  - include\sys_defs.h
  - include\sys_gpio.h
  - include\sys_i2c.h
  - include\sys_low_power.h
  - include\sys_spi.h
  - include\sys_task.h
  - include\sys_uart1.h
  - sys_config.c
  - sys_gpio.c
  - sys_i2c.c
  - sys_low_power.c
  - sys_spi.c
  - sys_task.c
  - sys_uart1.c

工程编译后的固件：
.\ide\stone1.0\sdk\Debug\sdk.bin

如果需要与 Bootloader 合并烧录，输出的固件为：
.\ide\stone1.0\sdk\Debug\sdk.full.bin

3.5 PD 库

PD（board support package）是一个 **静态库**，所有 PD 应用程序都应该要链接 PD 库。

PD 库存放目录在：

.\firmware\stone1.0\pd\

PD 库由头文件和静态库（libpd.a）组成。其中头文件包含如下：

- dpm_cc.h：Type-C 的接口逻辑控制
- dpm_port.h：设备策略管理
- dpm_qc.h：高通快充协议实现
- pe_state.h：PD 策略引擎与状态机定义
- prl_hr_state.h：PD 协议层 Hard Reset 状态机定义
- prl_msg_defs.h：PD 协议层消息定义
- prl_msg_parser.h：PD 协议层消息解析
- prl_rx_state.h：PD 协议层消息接收状态机定义
- prl_tcpm.h：PD 协议层 tcpm 定义
- prl_tx_state.h：PD 协议层消息发送状态机定义
- reg_def_pd.h：PD 模块寄存器定义
- usbpd_config.h：PD 模块配置项定义
- usbpd_defs.h：PD 模块定义
- usbpd_hook.h：PD 模块回调定义
- usbpd_main.h：PD 模块入口
- usbpd_timer_defs.h：PD 模块定时器定义
- usbpd.h：PD 模块全局头文件，该文件已包含上述所列头文件。

3.6 pd.drp

pd.drp 工程是 UM3506 SoC 的一个完整的 PD 应用例子。

目前支持功能：

- CLI 命令行接口
- 所有外设测试接口
- 系统配置保存
- 低功耗模式
- Type-C CC 配置
- PD DRP 模式
- PWM 控制 Source 供电
- OLED 驱屏例子

代码包含：

- main.c：程序入口实现，链接到.\firmware\stone1.0\pd.drp\main.c
- platform_config.h：配置文件，链接到.\firmware\stone1.0\pd.drp\platform_config.h
- cli：cli 子目录，链接到.\firmware\stone1.0\sdk\cli 目录
  - include\cli_cmd_base.h
  - include\cli_cmd_usbpd.h：
    链接到.\firmware\stone1.0\pd.drp\cli\include\cli_cmd_usbpd.h
  - include\cli_engine.h
  - cli_cmd_base.c
  - cli_cmd_usbpd.c：链接到.\firmware\stone1.0\pd.drp\cli\cli_cmd_usbpd.c
  - cli_engine.c
- system：system 子目录，链接到.\firmware\stone1.0\sdk\system 目录
  - include\sys_config.h

- include\sys_defs.h：
  链接到.\firmware\stone1.0\pd.drp\system\include\sys_defs.h
- include\sys_gpio.h：
  链接到.\firmware\stone1.0\pd.drp\system\include\sys_gpio.h
- include\sys_i2c.h
- include\sys_low_power.h
- include\sys_oled_display.h：
  链接到.\firmware\stone1.0\pd.drp\system\include\sys_oled_display.h
- include\sys_pd_config.h：
  链接到.\firmware\stone1.0\pd.drp\system\include\sys_pd_config.h
- include\sys_pd_hook.h：
  链接到.\firmware\stone1.0\pd.drp\system\include\sys_pd_hook.h
- include\sys_spi.h
- include\sys_task.h
- include\sys_uart1.h
- sys_config.c
- sys_gpio.c：链接到.\firmware\stone1.0\pd.drp\system\sys_gpio.c
- sys_i2c.c
- sys_low_power.c
- sys_oled_display.c：
  链接到.\firmware\stone1.0\pd.drp\system\sys_oled_display.c
- sys_pd_config.c：链接到.\firmware\stone1.0\pd.drp\system\sys_pd_config.c
- sys_pd_hook.c：链接到.\firmware\stone1.0\pd.drp\system\sys_pd_hook.c
- sys_spi.c
- sys_task.c
- sys_uart1.c

注意：

■ PD 功能需要链接 libpd.a，存放在 firmware\stone1.0\pd 目录下

工程编译后的固件：
.\ide\stone1.0\pd.drp\Debug\pd.drp.bin

如果需要与 Bootloader 合并烧录，输出的固件为：
.\ide\stone1.0\pd.drp\Debug\pd.drp.full.bin

3.7 pd.pb.1c2a

pd.pb.1c2a 工程是 UM3506 SoC 的一个完整的 1C2A PD 移动电源应用例子。
目前支持功能：

- CLI 命令行接口
- 系统配置保存
- 低功耗模式
- Type-C CC 配置
- PD DRP 模式
- USB-A 支持 QC 模式
- SC8812A 电源芯片
- 支持 100W 充放电
- 支持功率控制
- LED 电量显示

- 过流保护

…

详细功能定义请看文档：
C:\uwe-ide-win\workspace\firmware\stone1.0\pd.pb.1c2a\docs\UM3506 DEMO.xlsx

代码包含：

- main.c：程序入口实现，链接到.\firmware\stone1.0\pd.pb.1c2a\main.c
- platform_config.h：配置文件，链接到.\firmware\stone1.0\pd.pb.1c2a\platform_config.h
- cli：cli 子目录，链接到.\firmware\stone1.0\sdk\cli 目录
  - include\cli_cmd_base.h
  - include\cli_cmd_usbpd.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\cli\include\cli_cmd_usbpd.h
  - include\cli_engine.h
  - cli_cmd_base.c
  - cli_cmd_usbpd.c：链接到.\firmware\stone1.0\pd.pb.1c2a\cli\cli_cmd_usbpd.c
  - cli_engine.c
- system：system 子目录，链接到.\firmware\stone1.0\sdk\system 目录
  - include\sc8812_reg_defs.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sc8812_reg_defs.h
  - include\sc8815_reg_defs.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sc8815_reg_defs.h
  - include\sys_bat_intf.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_bat_intf.h
  - include\sys_config.h
  - include\sys_defs.h：链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_defs.h
  - include\sys_gpio.h：链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_gpio.h
  - include\sys_i2c.h
  - include\sys_low_power.h
  - include\sys_pd_config.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_pd_config.h
  - include\sys_pd_hook.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_pd_hook.h
  - include\sys_qc_hook.h：
    链接到.\firmware\stone1.0\pd.pb.1c2a\system\include\sys_qc_hook.h
  - include\sys_spi.h
  - include\sys_task.h
  - include\sys_uart1.h
  - sys_bat_intf.c：链接到.\firmware\stone1.0\pd.pb.1c2a\system\sys_bat_intf.c
  - sys_config.c
  - sys_gpio.c：链接到.\firmware\stone1.0\pd.pb.1c2a\system\sys_gpio.c
  - sys_i2c.c
  - sys_low_power.c
  - sys_pd_config.c：链接到.\firmware\stone1.0\pd.pb.1c2a\system\sys_pd_config.c
  - sys_pd_hook.c：链接到.\firmware\stone1.0\pd.pb.1c2a\system\sys_pd_hook.c
  - sys_qc_hook.c：链接到.\firmware\stone1.0\pd.pb.1c2a\system\sys_qc_hook.c
  - sys_spi.c
  - sys_task.c
  - sys_uart1.c

注意：

- PD 和 QC 功能需要链接 libpd.a，存放在 firmware\stone1.0\pd 目录下

工程编译后的固件：
.\ide\stone1.0\pd.pb.1c2a\Debug\pd.pb.1c2a.bin

如果需要与 Bootloader 合并烧录，输出的固件为：
.\ide\stone1.0\pd.pb.1c2a\Debug\pd.pb.1c2a.full.bin

3.8 pd.snk

pd.snk 工程是 UM3506 SoC 的一个 PD Sink 的应用例子。
目前支持功能：

- CLI 命令行接口
- 所有外设测试接口
- 系统配置保存
- 低功耗模式
- Type-C CC 配置
- PD Sink 模式
- 18W 的操作功率，最大吸收电压 9V

代码包含：

- main.c：程序入口实现，链接到.\firmware\stone1.0\pd.snk\main.c
- platform_config.h：配置文件，链接到.\firmware\stone1.0\pd.snk\platform_config.h
- cli：cli 子目录，链接到.\firmware\stone1.0\sdk\cli 目录
  - include\cli_cmd_base.h
  - include\cli_cmd_usbpd.h：链接到.\firmware\stone1.0\pd.snk\cli\include\cli_cmd_usbpd.h
  - include\cli_engine.h
  - cli_cmd_base.c
  - cli_cmd_usbpd.c：链接到.\firmware\stone1.0\pd.snk\cli\cli_cmd_usbpd.c
  - cli_engine.c
- system：system 子目录，链接到.\firmware\stone1.0\sdk\system 目录
  - include\sys_config.h
  - include\sys_defs.h：链接到.\firmware\stone1.0\pd.snk\system\include\sys_defs.h
  - include\sys_gpio.h：链接到.\firmware\stone1.0\pd.snk\system\include\sys_gpio.h
  - include\sys_i2c.h
  - include\sys_low_power.h
  - include\sys_pd_config.h：链接到.\firmware\stone1.0\pd.snk\system\include\sys_pd_config.h
  - include\sys_pd_hook.h：链接到.\firmware\stone1.0\pd.snk\system\include\sys_pd_hook.h
  - include\sys_qc_hook.h：链接到.\firmware\stone1.0\pd.snk\system\include\sys_qc_hook.h
  - include\sys_spi.h
  - include\sys_task.h
  - include\sys_uart1.h
  - sys_config.c
  - sys_gpio.c：链接到.\firmware\stone1.0\pd.snk\system\sys_gpio.c
  - sys_i2c.c
  - sys_low_power.c

- sys_pd_config.c：链接到.\firmware\stone1.0\pd.snk\system\sys_pd_config.c
- sys_pd_hook.c：链接到.\firmware\stone1.0\pd.snk\system\sys_pd_hook.c
- sys_qc_hook.c：链接到.\firmware\stone1.0\pd.snk\system\sys_qc_hook.c
- sys_spi.c
- sys_task.c
- sys_uart1.c

注意：

■ PD 功能需要链接 libpd.a，存放在 firmware\stone1.0\pd 目录下

工程编译后的固件：
.\ide\stone1.0\pd.snk\Debug\pd.snk.bin

如果需要与 Bootloader 合并烧录，输出的固件为：
.\ide\stone1.0\pd.snk\Debug\pd.snk.full.bin

4 BSP 库

4.1 硬件接口

4.1.1 系统实时时钟

系统实时时钟是由 CPU 内核的内部定时器产生，SDK 进行了封装，内部定时器初始化为按每 1 微秒计数一次的时钟精度。

定义在 drv_rtc.h，相关函数如下：

4.1.1.1 系统时钟初始化函数-void scr_rtc_init(void)

功能	对时钟控制器（01），定时器比较器，定时器计数器，定时器分频寄存器进行初始化
定义	void scr_rtc_init(void)
参数	无
返回	无

4.1.1.2 时钟转换为 ms 时间函数- long ticks2ms(sys_tick_t t)

功能	将一个时钟数转换成 ms 时间
函数定义	long ticks2ms(sys_tick_t t)
参数	参数 t 是一个时钟数，长度为无符号型 64 位的整数
返回	返回 t 转换的 ms 时间，类型 long

4.1.1.3 ms 时间转换为时函数-sys_tick_t ms2ticks(long t)

功能	将一个 ms 数转换成时钟数
函数定义	sys_tick_t ms2ticks(long t)
参数	参数 t 为 ms 单位时间
返回	返回 t 转换后的时钟数

4.1.1.4 系统时钟延时函数-void rtc_delay_us(unsigned us)

功能	延时 us 微秒
函数定义	void rtc_delay_us(unsigned us)
参数	参数 us 延时的时间
返回	无

4.1.1.5 设置系统计数器比较值-void scr_rtc_setcmp(uint64_t when)

功能	设置计数器比较寄存器为传入的参数值
函数定义	void scr_rtc_setcmp(uint64_t when)
参数	参数 when 为要比较的时间长度
返回	无

4.1.1.6 系统定时器中断使能函数- SCR_RTC_TIMER_ENABLE()

功能	定时器中断使能函数
函数定义	SCR_RTC_TIMER_ENABLE()
参数	无
返回	无

4.1.1.7 系统定时器中断关闭函数-SCR_RTC_TIMER_DISABLE（）

功能	定时器中断关闭函数
函数定义	SCR_RTC_TIMER_DISABLE
参数	无
返回	无

4.1.1.8 获取系统上电到现在的时间

功能	系统上电后实时的时间，单位 us 秒
函数定义	now()
参数	无
返回	系统上电到现在的时间

4.1.2 ADC 接口

ADC 接口函数位于 Drv_adc .h ,其中 adc 的精度为 12 位

4.1.2.1 设置 ADC 采样时间 void drv_adc_stc_set(u32_t stc)

功能	以时钟周期为单位设置 ADC 采样时间，寄存器为 0x40000034 的 4~6 位
函数定义	drv_adc_stc_set(u32_t stc);
参数	参数的 stc 代表 adc 采样时间选择，范围 0~7
返回	无

4.1.2.2 获取 adc 采样时间-drv_adc_stc_get()

功能	获取 adc 采样时间，单位是一个时钟周期
函数定义	drv_adc_stc_get()
参数	无
返回	无

4.1.2.3 读一个通道 ADC 的值 u32_t drv_adc_read(u32_t channel);

功能	读一次输入通道的 adc 值
函数定义	u32_t drv_adc_read(u32_t channel)
参数	参数 channel 代表要采样的 adc 通道
返回	返回 12 位 ADC 结果

4.1.2.4 重复 count 次读取指定通道的 adc 的值

功能	重复 count 次读取指定通道的 adc 的值
函数定义	u32_t drv_adc_read_multi(u32_t channel, u32_t count);
参数	channel 为要读取的 adc 通道 ， count 是重复读取的次数 0~255
返回	返回 12 位 ADC 结果

4.1.2.5 连续读取指定 ADC 通道 count 次并累加后取平均值

功能	连续读取指定 ADC 通道 count 次并累加后取平均值
函数定义	u32_t drv_adc_read_multi_avg(u32_t channel, u32_t count);
参数	Channel 为要读取的 adc 通道，count 为读取 adc 结果寄存器的次数，其中最多读取 1024 次
返回	返回平均值

4.1.2.6 获取 adc 值的高 8 位-drv_adc_8bits_data_get(input)

功能	获取高 8 位 adc 的值
函数定义	drv_adc_8bits_data_get(input)
参数	参数 input 一般为一个 adc 值
返回	返回平均值

4.1.2.7 初始化电压增益-void drv_adc_current_sense_init(drv_adc_igain_t gain)

功能	初始化电压增益
函数定义	void drv_adc_current_sense_init(drv_adc_igain_t gain)
参数	参数 gain 为枚举类型，有两种输入 0，1；输入 0 为提高 adc 增益为 14，输入 1 提高 adc 增益为 15
返回	无

4.1.2.8 获取电压增益类型-drv_adc_igain_t drv_adc_current_sense_gain_get()

功能	获取电压增益类型
函数定义	drv_adc_igain_t drv_adc_current_sense_gain_get()
参数	无
返回	返回一个枚举类型值 0 或者 1

4.1.2.9 获取 adc 电压增益倍数 -u32_t drv_adc_current_sense_gain_times_get()

功能	Adc 获取电压增益倍数
函数定义	u32_t drv_adc_current_sense_gain_times_get()
参数	无
返回	返回 14 或者 15

4.1.2.10 获取 ibus 的电流

功能	获取 ibus 的电流
函数定义	u32_t drv_adc_ibus_get(u32_t sample_resister)
参数	参数 sample_resister 为输入的电阻值
返回	返回 ibus 的电流值

4.1.3 CRC 校验

该接口位于 SDK1.0/bsp/include/drivers.h，函数文件为 Drv_crc.c 和 Drv_crc.h

4.1.3.1 设置数据输入方向-drv_crc_bit_order_set(lsb)

功能	设置数据是先输入数据的 lsb 位还是 msb 位，确定数据的输入方向
函数定义	drv_crc_bit_order_set(lsb)
参数常量	参数 lsb，输入 1，数据 lsb 位先输入；输入 0，数据 msb 位先输入
返回	无

4.1.3.2 设置 crc 校验模式-drv_crc_mode_set(mode)

功能	设置 crc 校验码模式
函数定义	drv_crc_mode_set(mode)
参数常量	参数 mode， 输入 00 保留；输入 01，8 位 crc 校验码 mode；输入 10，16 位 crc 校验码 mode；输入 11，32 位 crc 校验码 mode
返回	无

4.1.3.3 设置 crc 校验码初始值-drv_crc_initial_set(value)

功能	设置 crc 校验码的初始值
函数定义	drv_crc_initial_set(value)
参数常量	参数 value，一般为 0 或者 0xF..F
返回	无

4.1.3.4**计算crc-drv_crc_input_data(value)**

功能	输入数据，去计算 crc
函数定义	drv_crc_input_data(value)
参数常量	参数 value 为输入的数据，一般为 8bit 数据
返回	无

4.1.3.5**获取8位crc校验码-u8_t drv_crc8(const u8_t *buf, int len)**

功能	获取输入数据的 8 位 crc 校验码
函数定义	u8_t drv_crc8(const u8_t *buf, int len)
参数常量	参数 buf，为输入的数据，len 数据的长度
返回	返回 8 位的 crc 校验码

4.1.3.6 获取16位crc校验码-u8_t drv_crc16(const u8_t *buf, int len)

功能	获取数据数据段的 16 位 crc 校验码
函数定义	u8_t drv_crc16(const u8_t *buf, int len)
参数常量	参数 buf，为输入的数据，len 数据的长度
返回	返回 16 位 crc 校验码

4.1.3.7 **获取32位校验码-u8_t drv_crc32(const u8_t *buf, int len)**

功能	获取数据数据段的 32 位 crc 校验码
函数定义	u8_t drv_crc32(const u8_t *buf, int len)
参数常量	参数 buf，为输入的数据，len 数据的长度
返回	返回 32 位 crc 校验码

4.1.4 DMA 操作

函数文件为 Drv_crc.c 和 Drv_crc.h

4.1.4.1 DMA0 内存拷贝

功能	DMA0 内存拷贝
函数定义	void *drv_dma_memcpy(void *dst, const void *src, u32_t count, DRV_DMA_MODE_t mode);
参数常量	*dst=目标地址指针；*src=源地址指针；count=数量；mode=0~3（0=增长模式，1=下降模式，2=源地址保持，3=目标地址保持）；

返回	无

4.1.4.2 DMA0 内存移动

功能	DMA0 内存移动
函数定义	void *drv_dma_memmove(void *dst, const void *src, u32_t count);
参数常量	*dst 目标地址、*src 源地址、count 拷贝数量
返回	无

4.1.4.3 DMA1 内存副本与中断支持

功能	DMA0 内存移动
函数定义	void drv_dma1_memcpy(void *dst, const void *src, u32_t count, DRV_DMA_MODE_t mode, bool_t intr, v_func_ptr_t intr_cb);
参数常量	DRV_DMA_MODE_t mode 模式、bool_t intr 是否中断使能、v_func_ptr_t intr_cb 如果使能中断是否返回函数功能名
返回	无

4.1.4.4 停止 DMA1 内存操作

功能	短暂停止 dma1 内存操作
函数定义	void drv_dma1_stop(void);
参数常量	无
返回	无

4.1.4.5 获取 dm0 目标数据初始值-drv_dma0_dst_addr_get()

功能	获取 dma0 目标数据初始地址
函数定义	drv_dma0_dst_addr_get()
参数常量	无
返回	无

4.1.4.6 获取源数据地址-drv_dma0_src_addr_get()

功能	获取 dma0 源数据地址
函数定义	drv_dma0_src_addr_get()
参数常量	无
返回	无

4.1.4.7 获取 dma1 目标数据地址-drv_dma1_dst_addr_get()

功能	获取 dma1 目标数据地址
函数定义	drv_dma1_dst_addr_get()

参数常量	无
返回	无

4.1.4.8 drv_dma1_src_addr_get()

功能	获取 dma1 原始数据地址
函数定义	drv_dma1_src_addr_get()
参数常量	无
返回	无

4.1.5 Flash 存储

代码文件 Drv_flash.c

4.1.5.1 写 32 比特位无符号整形到闪存

功能	写 32 比特位无符号整形到闪存
函数定义	bool_t drv_flash_write_32bits(u32_t addr, u32_t input);
参数常量	Addr=开始地址，input=输入一个 32 位数据
返回	True 或者 false

4.1.5.2 flash 扇区擦除 void drv_flash_sector_erase(u32_t addr)

功能	擦除4kb的数据，将扇区数据数据设置为0xFFFFFFFF
函数定义	void drv_flash_sector_erase(u32_t addr);
参数常量	Addr=开始地址，地址必须在 4096 字节边界对齐，并固定擦除一个扇区的数据
返回	无

4.1.5.3 通过 DMA 进行的 Flash 读操作 drv_flash_read

功能	通过硬件 DMA 进行简单的 Flash 读取
函数定义	void drv_flash_read(u32_t addr, u8_t *buf, int len);
参数常量	Addr 是所读地址，*buf 数据读取存放空间，len 指读取的数据长度
返回	无

4.1.5.4 通过硬件 DMA 进行简单的 Flash 写操作 drv_flash_write

功能	通过硬件DMA进行简单的Flash写操作
函数定义	bool_t drv_flash_write(u32_t addr, u8_t *buf, int len);
参数常量	Addr 开始写入地址，*buf 指向目标地址，len 写入字节长度
返回	true/false

4.1.5.5 通过软件拷贝进行简单的 Flash 读取 drv_flash_read2

功能	通过软件拷贝进行简单的Flash读取
函数定义	void drv_flash_read2(u32_t addr, u8_t *buf, int len);
参数常量	Addr 开始读入地址;*buf 为指向复制过来的目标地址;len 读入长度
返回	True/false

4.1.5.6 通过软件拷贝进行简单的 Flash 写操作 drv_flash_write2

功能	通过软件拷贝进行简单的Flash写操作
函数定义	bool_t drv_flash_write2(u32_t addr, u8_t *buf, int len);
参数常量	Addr 开始读入地址;*buf 为指向复制过来的目标地址;len 读入长度
返回	True/false

4.1.5.7 启用或者禁用安全寄存器操作 drv_flash_nvr_en

功能	启用或者禁用安全寄存器操作
函数定义	void drv_flash_nvr_en(bool_t en)
参数常量	参数 en ：0 或者 1 输入
返回	无

4.1.5.8 硬件电压相关初始化以及设置

功能	硬件电压相关初始化
函数定义	void trim_code_init(void)
参数	无
返回	无
功能	获取电流感应偏移
函数定义	u32_t drv_flash_trim_current_sense_offset_get()
参数常量	无
返回	32 位电流偏移量

功能	设置电流感应偏移
函数定义	void drv_flash_trim_current_sense_offset_set(u32_t offset)
参数常量	Offset：电流偏移量
返回	无

功能	获取参考电压
函数定义	u32_t drv_flash_trim_ref_volt_get()
参数常量	无

返回	返回无符号 32 位参考电压值

功能	设置参考电压
函数定义	void drv_flash_trim_ref_volt_set(u32_t ref_volt)
参数常量	参数 ref_volt 是指要设置的电压值
返回	无

4.1.6 GPIO 接口

4.1.6.1 gpio 相关操作

功能	获取对应的gpio口的使能状态
函数定义	drv_gpio_en_get(id)
参数常量	Id：gpio 端口值
返回	0 或者 1

4.1.6.2 设置 gpio 的使能状态

功能	设置对应的gpio使能状态
函数定义	drv_gpio_en_set(id,flg)
参数常量	Id：gpio 端口值 flg：0 或者 1
返回	无

4.1.6.3 获取 gpio 输入输出方向

功能	获取gpio输入输出方向，0为输入，1为输出
函数定义	drv_gpio_en_get(id)
参数常量	Id：gpio 端口值
返回	0 或者 1

4.1.6.4 设置 gpio 输出状态

功能	设置对应的gpio口的为输出状态，1为输出 0为输入
函数定义	drv_gpio_oe_set(id,flg)
参数常量	Id：gpio 端口值 flg：1 或者 0
返回	无

4.1.6.5 获取 gpio 口的下拉电阻状态

功能	获取gpio的下拉状态，
函数定义	drv_gpio_pd_get(id)
参数常量	Id：gpio 端口值
返回	1 或者 0， 1 为有下拉电阻，0 为没有下拉电阻

4.1.6.6 设置 gpio 的下拉电阻状态

功能	设置gpio下拉状态，

函数定义	drv_gpio_pd_set(id,flg)
参数常量	Id：gpio 端口值；flg ：1 或者 0 ，1 为使能状态，0 为关闭
返回	无

4.1.6.7 获取 gpio 上拉电阻状态

功能	获取gpio口的上拉状态
函数定义	drv_gpio_pu_get(id)
参数常量	Id：gpio 端口值
返回	0 或者 1，1 有上拉电阻，0 没有上拉

4.1.6.8 设置 gpio 的上拉电阻状态

功能	设置gpio的上拉状态
函数定义	drv_gpio_pu_set(id,flg)
参数常量	Id：gpio 端口值；flg：1 使能上拉，0 关闭上拉
返回	无

4.1.6.9 设置 gpio 的初始电平

功能	设置gpio的初始电平
函数定义	drv_gpio_val_set(id,flg)
参数常量	Id：gpio 端口值；flg： 1 高电平，0 低电平
返回	无

4.1.6.10 获取 gpio 中断的使能状态

功能	获取gpio中断的使能状态
函数定义	drv_gpio_intr_en_get(id)
参数常量	Id：gpio 端口值
返回	1 使能，0 关闭

4.1.6.11 获取 gpio 中断屏蔽状态

功能	获取gpio 中断是否屏蔽
函数定义	drv_gpio_intr_mask_get(id)
参数常量	Id：gpio 端口值
返回	1 打开屏蔽，0 无

4.1.6.12 获取 gpio 中断的触发类型

功能	获取 gpio 中断的触发类型
函数定义	drv_gpio_intr_type_get(id)
参数常量	Id：gpio 端口值
返回	0 电平触发，1 边沿触发

4.1.6.13 设置 gpio 中断的触发类型

功能	设置gpio中断的触发类型
函数定义	drv_gpio_intr_type_set(id,flg)
参数常量	Id：gpio 端口值；flg： 1 边沿触发，0 电平触发
返回	无

4.1.6.14 设置 gpio 中断的触发条件

功能	设置gpio中断的触发条件
函数定义	drv_gpio_intr_polar_set(id, flg)
参数常量	Id：gpio 端口值；flg： 1 高电平或者上升沿触发，0 低电平或者下降沿
返回	无

4.1.6.15 获取中断 gpio 的触发条件

功能	获取中断gpio的触发条件
函数定义	drv_gpio_intr_polar_get(id)
参数常量	Id：gpio 端口值
返回	1 高电平或者上升沿触发，0 低电平或者下降沿触发

4.1.6.16 获取 gpio 的屏蔽状态

功能	获取gpio的屏蔽状态
函数定义	drv_gpio_bit_mask_get(id)
参数常量	Id：gpio 端口值；flg
返回	1 屏蔽 gpio 输出写入；0 无屏蔽

4.1.6.17 设置 gpio 的屏蔽状态

功能	设置gpio的屏蔽状态
函数定义	drv_gpio_bit_mask_set(id,flg)
参数常量	Id：gpio 端口值；flg： 1 屏蔽，0 不屏蔽
返回	无

4.1.6.18 获取 gpio 中断的原始状态

功能	获取gpio中断的原始状态
函数定义	drv_gpio_intr_sta_get(id)
参数常量	Id：gpio 端口值
返回	0 或者 1

4.1.6.19 获取屏蔽中断的状态

功能	设置gpio的屏蔽状态
函数定义	drv_gpio_intr_sta_masked_get(id)
参数常量	Id：gpio 端口值

返回	0 或者 1

4.1.6.20 获取 gpio pad 施密特缓冲器使能状态

功能	获取gpio pad 施密特缓冲器使能状态
函数定义	drv_gpio_sen_get(id)
参数常量	Id：gpio 端口值
返回	0 关闭 1 使能

4.1.6.21 设置 gpio pad 施密特缓冲器使能状态

功能	设置 gpio pad 施密特缓冲器使能状态
函数定义	drv_gpio_sen_set(id,flg)
参数常量	Id：gpio 端口值，flg：1 使能 0 关闭
返回	无

4.1.6.22 获取 gpio 电平状态

功能	获取gpio某端口输入值函数
函数定义	int drv_gpio_val_get(int id);
参数常量	Id=端口值
返回	高电平 1 或者低电平 0

4.1.6.23 启用/禁用 gpio 中断

功能	启用/禁用gpio中断
函数定义	void drv_gpio_intr_en_set(int id, int flg);
参数常量	Id=端口序号；flg=Enable(1) or disable（0）
返回	无

4.1.6.24 设置 gpio 中断的屏蔽状态

功能	屏蔽gpio中断
函数定义	void drv_gpio_intr_mask_set(int id, int flg);
参数常量	Id=端口号；flg= 1 屏蔽 或者 0 无屏蔽
返回	无

4.1.6.25 初始化 gpio 中断处理函数

功能	初始化gpio中断处理函数
函数定义	void drv_gpio_intr_handler_init(void);
参数常量	无

返回	无

4.1.6.26 gpio 中断使能

功能	gpio 中断使能
函数定义	void drv_gpio_intr_init(bool_t en);
参数常量	参数 en 1 使能，0 关闭
返回	无

4.1.6.27 设置 gpio 中断的回调函数

功能	**设置中断 gpio 处理函数**
函数定义	void drv_gpio_intr_handler_set(int id, v_func_ptr_t intr_func);
参数常量	Id=gpio 端口号；intr_func 对应的中断处理函数
返回	无

4.1.6.28 获取 gpio open-drain 的模式

功能	获取 gpio open-drain 的模式
函数定义	DRV_GPIO_OD_TYPE_t drv_gpio_od_get(u32_t id)
参数常量	参数 id ： gpio 端口号
返回	00 ， 01 ， 10 ， 11 四种模式

4.1.6.29 设置 gpio 的 open-drain 模式

功能	设置 gpio 的 open-drain 模式
函数定义	void drv_gpio_od_set(u32_t id, DRV_GPIO_OD_TYPE_t flg)
参数常量	参数 id： gpio 端口号，参数 fig 00 01 10 11 四种输入
返回	无

4.1.6.30 gpio 中断处理函数接口

功能	gpio 中断处理函数接口，根据对应的 gpio，执行对应的 handler
函数定义	void drv_gpio_isr(u32_t arg)
参数常量	参数 arg =gpio 口
返回	无

4.1.6.31 设置 gpio 中断回调带参函数

功能	设置 gpio 中断回调带参函数
函数定义	void drv_gpio_intr_handler_with_arg_set(int id, v_func_ptr_t intr_func, u32_t arg)
参数常量	Id=gpio 口；intr_func=gpio 处理函数；arg=传入的参数

返回	无

4.1.7 GPIO 接口捕获模式

Gpio 口的捕获模式，为了抓取 gpio 输入信号沿到沿的时间，相关函数在 Drv_gpio_capture.c 文件

4.1.7.1 获取指定捕获通道的目标 gpio 口

功能	捕获模式下，用于获取捕获的 gpio
函数定义	u32_t drv_cap_en_gpio_get(DRV_CAPTURE_ID_t id);
参数常量	Id：捕获通道
返回	捕获通道使能，返回 gpio 端口号；捕获通道没有使能，返回 32

4.1.7.2 获取指定的捕获通道的使能状态

功能	获取捕获通道是否使能
函数定义	bool_t drv_gpio_cap_en_get(DRV_CAPTURE_ID_t id);
参数常量	Id：捕获通道
返回	返回 1 使能；0 关闭

4.1.7.3 gpio 捕获功能设置

功能	关闭或者启用 gpio 捕获功能
函数定义	void drv_gpio_cap_en_set(DRV_CAPTURE_ID_t id, u32_t gpio_id, bool_t flg);
参数常量	Id=捕获通道，0 或者 1；gpio_id =捕获的目标 gpio；flg=标志位，启用 1，关闭 0
返回	无

4.1.7.4 配置 gpio 捕获模式

功能	配置 gpio 捕获模式
函数定义	void drv_gpio_cap_mode_set(DRV_CAPTURE_ID_t id, int mode);
参数常量	Id=捕获通道，0 或者 1；mode=捕获模式，0 两个上升沿之间的宽度，1 两个下降沿之间的宽度，2 上升沿到下降沿的宽度，3 下降沿到上升沿的宽度
返回	无

4.1.7.5 获取捕获通道的工作模式

功能	获取捕获通道的工作模式

函数定义	int drv_gpio_cap_mode_get(DRV_CAPTURE_ID_t id);
参数常量	Id=捕获通道
返回	返回 0 两个上升沿之间的宽度，返回 1 两个下降沿之间的宽度，返回 2 上升沿到下降沿的宽度，返回 3 下降沿到上升沿的宽度

4.1.7.6 获取捕获结果

功能	获取捕获结果
函数定义	u32_t drv_gpio_cap_count_get(DRV_CAPTURE_ID_t id);
参数常量	Id=捕获通道
返回	无符号 32 位时间宽度

4.1.8 I2C 接口驱动

主要实现 i2c 的初始化，i2c 配置，i2c 读写等函数

4.1.8.1 I2c 结构体初始化

功能	初始化结构体数组 g_i2c_info ，将其成员全部置 0
函数定义	void drv_i2c_init(void)
参数常量	无
返回	无

4.1.8.2 I2c 接口初始化

功能	初始化 i2c 接口
函数定义	static void drv_i2c_intf_init(DRV_I2C_ID_t i2c_id)
参数常量	i2c_id ： i2c 索引号，0 或者 1
返回	无

4.1.8.3 I2c 中断处理函数

功能	I2c 中断处理函数，实现数据的发送和接收
函数定义	void drv_i2c_intr_handler(u32_t arg)
参数常量	arg：触发中断的 i2c 中断号
返回	无

4.1.8.4 主机建立消息传递起始条件

功能	作为主机，建立 i2c 传输的起始信号
函数定义	static void drv_i2c_master_start(DRV_I2C_ID_t i2c_id)

参数常量	i2c_id： i2c 索引号，0 或者 1
返回	无

4.1.8.5 设置 i2c 的传输速率

功能	设置 i2c 传输速率
函数定义	void drv_i2c_master_freq_set(DRV_I2C_ID_t i2c_id, DRV_I2C_FREQ_t freq)
参数常量	i2c_id=i2c 索引号 ； freq=传输速率，枚举类型
返回	无

4.1.8.6 主机 I2C 数据传输接口函数

功能	作为主机时，实现对从机的数据读写操作
函数定义	u8_t drv_i2c_master_transmit(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t *ptx_data, u8_t tx_len, u8_t *prx_data, u8_t rx_len);
参数常量	i2c_id=i2c 索引号；addr =I2C 从设备地址；*ptx_data=发送数据地址指针；tx_len=发送数据长度；*prx_data=接受数据地址指针；rx_len=接收数据长度
返回	无符号 8 位的返回值，表示 i2c 传输状态

4.1.8.7 打开 i2c 功能

功能	设置从机地址，设置总线状态，以及中断初始化，添加回调函数据
函数定义	void drv_i2c_open(DRV_I2C_ID_t i2c_id, u8_t sla_addr)
参数常量	i2c_id=i2c 索引号 ； sla_addr=从机地址
返回	无

4.1.8.8 关闭 i2c 功能

功能	关闭 i2c 功能
函数定义	void drv_i2c_close(DRV_I2C_ID_t i2c_id)
参数常量	i2c_id=i2c 索引号
返回	无

4.1.8.9 设置从机地址

功能	作为从机时，设置我们自己地址
函数定义	void drv_i2c_sla_addr_set(DRV_I2C_ID_t i2c_id, u8_t addr)
参数常量	i2c_id=i2c 索引号 ； addr=从机地址

返回	无

4.1.8.10 设置从机接收函数

功能	设置从机接收函数
函数定义	void drv_i2c_sla_rx_handler_set(DRV_I2C_ID_t i2c_id, void (*sla_rx_func)(DRV_I2C_ID_t id, u8_t* received_data, u8_t received_len))
参数常量	i2c_id=i2c 索引号 ；*sla_rx_func=接收函数；*received_data=接受数据；received_len=接收数据长度
返回	无

4.1.8.11 设置从机发送函数

功能	设置从机发送函数
函数定义	void drv_i2c_sla_tx_handler_set(DRV_I2C_ID_t i2c_id, u8_t (*sla_tx_func)(DRV_I2C_ID_t id, u8_t tx_data_len_max, u8_t* transmit_data))
参数常量	i2c_id=i2c 索引号 ；*sla_tx_func=发送函数；*transmit_data=发送数据；tx_data_len_max=发送数据最大长度
返回	无

4.1.8.12 读取从机 8 位寄存器的值

功能	读取从机 8 位寄存器的值
功能	u8_t drv_i2c_reg_get(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg)
参数常量	i2c_id=i2c 索引号 ； addr=从机地址；reg=8 位寄存器地址
返回	返回 8 位数据

4.1.8.13 从机 8 位寄存器写值

功能	向一个从机的 8 位寄存器，写一个 8 位的数据
函数定义	void drv_i2c_reg_set(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t val)
参数常量	i2c_id=i2c 索引号 ； addr=从机地址；reg=8 位寄存器；val=8 位数据
返回	无

4.1.8.14 读取从机连续多个寄存器的值

功能	读取从机连续多个寄存器的值
函数定义	void drv_i2c_reg_mget(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t cnt, u8_t *p_val)
参数常量	i2c_id=i2c 索引号 ；addr=从机地址；reg=读取寄存器开始地址；cnt=读取数据长度；*p_val=读取的数据
返回	无

4.1.8.15 i2c 写入多个从机寄存器

功能	连续写入多个从机寄存器
函数定义	void drv_i2c_reg_mset(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t cnt, u8_t *p_val)
参数常量	i2c_id=i2c 索引号 ；addr=从机地址；reg=寄存器地址；cnt=写入数据的长度；*p_val=写入的数据
返回	无

4.1.8.16 i2c 读取从机寄存器若干 bit 的值

功能	读取从机寄存器 field 位置，width 宽度位的值
函数定义	u8_t drv_i2c_reg_field_get(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t width)
参数常量	i2c_id=i2c 索引号 ；addr=从机地址；reg=寄存器地址；field=位置；width=宽度
返回	返回无符号 8 位数据

4.1.8.17 设置从机寄存器存器若干 bit 的值

功能	配置从机寄存器 field 位置，width 宽度的值
函数定义	void drv_i2c_reg_field_set(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t width, u8_t val)
参数常量	i2c_id=i2c 索引号 ；addr=从机地址；reg=寄存器地址；field=位置；val=写入的数据
返回	无

4.1.8.18 i2c 读取寄存器，获取带掩码的数据

功能	通过 i2c 读取寄存器的值，屏蔽不需要的 bit 并返回
函数定义	u8_t drv_i2c_reg_field_get2(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t mask)
参数常量	i2c_id=i2c 索引号 ；addr=从机地址；reg=寄存器；field=位置；mask=屏蔽位
返回	返回无符号 8 位数据

4.1.8.19 使用掩码设置寄存器并取消屏蔽

功能	关闭 i2c 功能
函数定义	void drv_i2c_reg_field_set2(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t mask, u8_t unmask, u8_t val)
参数常量	i2c_id=i2c 索引号
返回	无

4.1.8.20 i2c 访问 16 位寄存器

u16_t drv_i2c_reg_get_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg)
void drv_i2c_reg_set_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u16_t val)
u16_t drv_i2c_reg_field_get_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t width)
void drv_i2c_reg_field_set_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u8_t width, u16_t val)
u16_t drv_i2c_reg_field_get2_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u16_t mask)
void drv_i2c_reg_field_set2_u16(DRV_I2C_ID_t i2c_id, u8_t addr, u8_t reg, u8_t field, u16_t mask, u16_t unmask, u16_t val)
I2c 读取 16 位寄存器的函数接口与 8 位寄存器的函数接口功能类似,详细请参照 4.1.8.12~4.1.8.19

4.1.9 中断

主要实现中断初始化，中断服务功能，中断类型有四种：系统内部中断，系统时钟中断，软件中断，外部中断。函数文件 Drv_interrupt.c 和 Drv_interrupt.h

4.1.9.1 中断初始化

功能	中断初始化
函数定义	void drv_intr_init(void);
参数常量	无
返回	无

4.1.9.2 获取外部中断状态

功能	获取外部中断状态
函数定义	bool_t drv_intr_ext_en_get(int idx);
参数常量	Idx=中断向量号
返回	0 或 1，不使能/使能

4.1.9.3 启用或禁用外部中断

功能	启用/禁用外部中断
函数定义	void drv_intr_ext_en_set(int idx, bool_t state);
参数常量	Idx=中断向量号；state=1 或者 0，使能或者禁止
返回	无

4.1.9.4 外部中断相关设置

功能	启用/禁用外部中断，设置中断优先级，设置中断处理函数
函数定义	void drv_intr_ext_set(int idx, bool_t state, int pri, v_func_ptr_t isr_func);

参数常量	Idx=中断向量号；state=1 或者 0；pri=优先等级 0～7；isr_func=中断处理函数
返回	无

4.1.9.5 外部中断初始化

功能	外部中断初始化，状态寄存器清 0，中断向量表和 handler 初始化为空
函数定义	static void drv_intr_ext_init(void)
参数常量	无
返回	无

4.1.9.6 内部中断服务

功能	内部异常处理并打印，内部中断处理
函数定义	int trap_handler(unsigned long cause, unsigned long epc, unsigned long *regs)
参数常量	Cause=异常编号；epc=异常中断号；*reg=程序运行寄存器地址
返回	运行正常返回 0；其他

4.1.9.7 软件中断服务

功能	处理软件中断
函数定义	int trap_int_msw_handler(unsigned long cause, unsigned long epc, unsigned long *regs)
参数常量	Cause=异常编号；epc=异常中断号；*reg=程序运行寄存器地址
返回	0

4.1.9.8 内部时钟中断服务

功能	内部时钟中断处理函数接口
函数定义	int trap_int_mtimer_handler(unsigned long cause, unsigned long epc, unsigned long *regs)
参数常量	Cause=异常编号；epc=异常中断号；*reg=程序运行寄存器地址
返回	0

4.1.9.9 外部中断服务程序

功能	外部中断处理函数接口
函数定义	int trap_int_mext_handler(unsigned long cause, unsigned long epc, unsigned long *regs)
参数常量	Cause=异常编号；epc=异常中断号；*reg=程序运行寄存器地址
返回	0

4.1.10 PWM

对 pwn 占空比，时钟周期等重新设置，必须重新对 pwn 通道进行使能操作才生效。

4.1.10.1 配置 PWM 初始值、PER 和比较值

功能	配置 PWM 初始值、PER 和比较值
函数定义	void drv_pwm_set(int id, bool_t init_high, u16_t per, u16_t cmp1, u16_t cmp2);
参数常量	Id=PWM 通道的序号（0~5）；init_high=初始电平；per=周期；cmp1=波形翻转点；cmp2=波形翻转点 2
返回	无

4.1.10.2 获取 pwn 通道使能状态

功能	获取 pwn 通道使能装填
函数定义	drv_pwm_en_get(id)
参数常量	Id= pwn 通道（0~5）
返回	1 使能或者 0 关闭

4.1.10.3 打开/关闭 pwn 功能

功能	Pwn 功能使能或者关闭
函数定义	drv_pwm_en_set(id,flg)
参数常量	Id= pwn 通道（0~5）；flg=1 或者 0，1 打开，0 关闭
返回	

4.1.10.4 设置 pwn 初始状态和波形周期

功能	Pwn 初始化设置，设置输出波形初始状态为高，还是低，设置波形周期
函数定义	drv_pwm_init_set(id,high,per)
参数常量	id=pwn 通道（0~5）；high=0 或 1 ，1 初始状态高，0 初始状态低；per=波形周期，决定 pwn 的频率
返回	无

4.1.10.5 获得 pwn 的波形初始状态

功能	获得 pwn 的波形初始状态
函数定义	drv_pwm_init_get(id)
参数常量	Id=pwn 通道 id（0~5）
返回	1 或 0

4.1.10.6 获取 pwn 波形周期

功能	获取 pwn 波形周期
函数定义	drv_pwm_per_get(id)
参数常量	Id=pwn 通道（0~5）
返回	时钟周期，16 位

4.1.10.7 占空比设置 1

功能	设置占空比
函数定义	void drv_pwm_duty_set(int id, u32_t cmp1, u32_t cmp2)
参数常量	Id=pwn 通道（0~5）；cmp1=比较点 1，当计数器达到 cmp1，转换电平输出；cmp2=比较点 2，当计数器达到 cmp2 时，输出再次切换
返回	无

4.1.10.8 占空比设置 2

功能	设置占空比，将比较点 1 和比较点 2 合并
函数定义	drv_pwm_duty_set2(id,dutyVal)
参数常量	Id=pwn 通道（0~5）；dutyVal=比较点，当计数器达到比较点，转换电平输出
返回	无

4.1.10.9 获取占空比

功能	获取占空比
函数定义	drv_pwm_duty_get(id)
参数常量	Id=pwn 通道（0~5）
返回	返回 32 位占空比

4.1.10.10 获取占空比比较点 1

功能	获取占空比比较点 1
函数定义	drv_pwm_duty_cmp1_get(id)
参数常量	Id=pwn 通道（0~5）
返回	返回比较点 1 的值，16 位

4.1.10.11 获取占空比比较点 2

功能	获取占空比比较点 2
函数定义	drv_pwm_duty_cmp2_get(id)
参数常量	Id=pwn 通道（0~5）
返回	返回比较点 1 的值，16 位

4.1.10.12 配置 pwn 初始值，波形周期，占空比

功能	Pwn 基本配置
函数定义	void drv_pwm_set(int id, bool_t init_high, u16_t per, u16_t cmp1, u16_t cmp2)
参数常量	Id=pwn 通道（0~5）；init_high=1 或 0，初始电平设置；per=波形周期；Cmp1=占空比比较点 1；cmp2=占空比比较点 2
返回	无

4.1.10.13 配置 pwn 初始值，波形周期，占空比 2

功能	配置 pwn 初始值，波形周期，占空比 2
函数定义	void drv_pwm_freq_set(int id, bool_t init_high, u32_t freq, u32_t duty);
参数常量	Id=pwn 通道（0~5）；init_high=电平初值；freq=设置频率；duty=占空比比值，最大 100
返回	无

4.1.10.14 pwn 两通道同时使能设置

功能	同时对 pwn 两通道进行打开关闭操作
函数定义	void drv_pwm_en_pair_set(int id1, int id2, bool_t flg);
参数常量	id1=pwn 通道（0~5）；id2=pwn 通道；flg=使能标志位，1 打开，0 关闭
返回	无

4.1.11 寄存器接口

寄存器设置，获取寄存器的值等操作，函数文件 Drv_reg_intf.c,Drv_reg_intf.h

4.1.11.1 获取寄存器的值

功能	获取寄存器的值
功能	u32_t drv_reg_get(u32_t reg); DRV_REG_GET(reg)
参数常量	Reg=寄存器地址
返回	32 位寄存器值

4.1.11.2 获取多个寄存器的值

功能	获取多个寄存器的值

函数定义	void drv_reg_mget(u32_t reg, u32_t cnt, u32_t *p_val); DRV_REG_MGET(reg,cnt,pval)
参数常量	Reg=寄存器地址；cnt=数量；p_val=获取寄存器值，指针类型
返回	无

4.1.11.3 设置寄存器的值

功能	设置寄存器的值
函数定义	void drv_reg_set(u32_t reg, u32_t val); DRV_REG_SET(reg,val)
参数常量	Reg=寄存器地址；val=要写入的寄存器值
返回	无

4.1.11.4 设置多个寄存器的值

功能	设置多个寄存器的值
函数定义	void drv_reg_mset(u32_t reg, u32_t cnt, u32_t *p_val); DRV_REG_MSET(reg,cnt,pval)
参数常量	Reg=寄存器地址；cnt=数量；p_val=目标写入值，指针类型
返回	无

4.1.11.5 获取某宽度的寄存器字段

功能	获取某宽度的寄存器字段
函数定义	u32_t drv_reg_field_get(u32_t reg, u32_t field, u32_t width);
参数常量	Reg=寄存器地址；field=偏移量；width=字段宽度
返回	返回 width 位的寄存器值

4.1.11.6 设置某宽度的寄存器字段

功能	设置某宽度的寄存器字段
函数定义	void drv_reg_field_set(u32_t reg, u32_t field, u32_t width, u32_t val);
参数常量	Reg=寄存器地址；field=偏移量；width=字段宽度；val=写入值
返回	无

4.1.11.7 获取带有掩码的寄存器字段

功能	获取带有掩码的寄存器字段

函数定义	u32_t drv_reg_field_get2(u32_t reg, u32_t field, u32_t field_set);
参数常量	Reg=寄存器地址；field=偏移量；field_set=掩码值
返回	返回寄存器字段的值

4.1.11.8 设置带有掩码的寄存器字段

功能	设置带有掩码的寄存器字段
函数定义	void drv_reg_field_set2(u32_t reg, u32_t field, u32_t field_set, u32_t field_mask, u32_t val);
参数常量	Reg=待获取寄存器；field=位域；field_set=掩码值；field_mask=屏蔽码；val=写入值
返回	无

4.1.11.9 获取某宽度的寄存器字段 2

功能	获取某宽度的寄存器字段
函数定义	DRV_REG_FIELD_GET2(reg,field,width)
参数常量	Reg=寄存器地址；field=宏定义；width=字段宽度
返回	返回 width 位的寄存器值

4.1.11.10 设置某宽度的寄存器字段 2

功能	设置某宽度的寄存器字段
函数定义	DRV_REG_FIELD_SET2(reg,field,width,val)
参数常量	Reg=寄存器地址；field=宏定义；width=字段宽度；val=写入值
返回	无

4.1.11.11 获取带有掩码的寄存器字段 2

功能	获取带有掩码的寄存器字段
函数定义	DRV_REG_FIELD_GET(reg,field)
参数常量	Reg=寄存器地址；field=宏定义
返回	返回寄存器字段的值

4.1.11.12 设置带有掩码的寄存器字段 2

功能	设置带有掩码的寄存器字段
函数定义	DRV_REG_FIELD_SET(reg,field,val)
参数常量	Reg=待获取寄存器；field=宏定义；val=写入值
返回	无

4.1.12 SPI 接口

函数文件 Drv_spi.c 和 Drv_spi.h

4.1.12.1 SPI 中断处理函数

功能	SPI 中断处理函数，接受或者发送数据
函数定义	void drv_spi_intr_handler(u32_t arg)
参数常量	Arg
返回	无

4.1.12.2 SPI 初始化配置

功能	初始化 spi 功能
函数定义	u8_t drv_spi_open(u8_t master, u8_t ssn, u8_t data_rate, u8_t cpol_cpha)
参数常量	Master=1 或者 0,1 主机模式,0 从机模式;ssn=spi 接口(0~3);data_rate=spi 传输速率；cpol_cpha=spi 通信模式
返回	0

4.1.12.3 关闭 SPI

功能	关闭 spi 功能
函数定义	void drv_spi_close(void)
参数常量	无
返回	无

4.1.12.4 SPI 的 DMA 传输启动

功能	SPI 的 DMA 传输启动
函数定义	void drv_spi_sla_dma_start(void)
参数常量	无
返回	无

4.1.12.5 冲洗缓存区

功能	从机模式冲洗缓存区
函数定义	void drv_spi_sla_flush(void)
参数常量	无
返回	无

4.1.12.6 spi 从机模式 DMA 中断处理

功能	spi 从机模式 DMA 中断处理，用于向主机发送数据
函数定义	`static void drv_spi_sla_dma_handler(u32_t arg)`
参数常量	Arg
返回	无

4.1.12.7 SPI 的主机模式发接口

功能	SPI 主机模式发送接口
函数定义	`u8_t drv_spi_transmit(u8_t *tx_buf, u8_t *rx_buf, int len)`
参数常量	tx_buf=发送的数据 buff；rx_buf=接受的数据 buff；len=数据长度
返回	无

4.1.12.8 spi 从机模式接口选择

功能	spi 从机模式接口选择
函数定义	`drv_spi_sla_sel_on(ssn)`
参数常量	Ssn=0~3
返回	无

4.1.12.9 spi 从机模式接口设置清除

功能	spi 从机模式接口设置清除
函数定义	drv_spi_sla_sel_off()
参数常量	无
返回	无

4.1.12.10 启动/禁用 spi 中断

功能	启动/禁用 spi 中断
函数定义	drv_spi_intr_en(); drv_spi_intr_disable()
参数常量	无
返回	无

4.1.12.11 获取 spi 中断的状态

功能	获取 spi 中断的状态
函数定义	drv_spi_en_get()
参数常量	无
返回	1（使能）或者 0（禁用）

4.1.12.12 spi 禁用/启用 ssn 输入

功能	spi 禁用/启用 ssn 输入

函数定义	drv_spi_ssdis_set(flg)
参数常量	Flg= 0 或者 1，0 启用 snn 输入，1 禁用 snn 输入
返回	无

4.1.12.13 spi 通信 ssn 输入使能状态获取

功能	spi 通信 ssn 输入使能状态获取
函数定义	drv_spi_ssdis_get()
参数常量	无
返回	1（禁用）或者 0（使能）

4.1.12.14 spi 通信传输一个字节

功能	spi 通信传输一个字节
函数定义	drv_spi_transmit_data(tx_data)
参数常量	tx_data=传输的数据 8bit
返回	无

4.1.12.15 spi 通信接收一个字节数据

功能	spi 通信接收一个字节数据
函数定义	drv_spi_get_data()
参数常量	无
返回	一个字节数据

4.1.12.16 spi 通信调试设置

功能	Spi debug 开关
函数定义	drv_spi_isr_dbg_set(en)
参数常量	en=1 或者 0，1 使能，0 关闭
返回	无

4.1.12.17 spi 调试开关状态获取

功能	获取 spi 是否处于 debug 状态
函数定义	drv_spi_isr_dbg_get()
参数常量	无
返回	1（打开）或者 0（关闭）

4.1.12.18 设置 spi 从机模式设置传输/接收的数据长度

功能	设置 spi 从机模式设置传输/接收的数据长度
函数定义	drv_spi_sla_rx_bytes_set(rxbytes) drv_spi_sla_tx_bytes_set(txbytes)
参数常量	Rxbytes=接收数据的长度；txbytes=发送数据的长度

返回	无

4.1.13 硬件扩展定时器

函数文件 Drv_timer.c 和 Drv_time.h

4.1.13.1 硬件扩展定时器初始化

功能	启用硬件扩展定时器 3，设置时钟 3 中断产生时间，设置处理函数
函数定义	void drv_timer_init(void)
参数常量	无
返回	无

4.1.13.2 启动定时器

功能	添加一个定时器到 driver_timer 并启动定时器
函数定义	bool_t drv_timer_start(u32_t timeout, volatile drv_timer_t *timer, void *cb)
参数常量	Timeout=定时器溢出时间；time=软件定时器对象；cb=定时回调处理函数
返回	1 启动成功或者 0 没有添加成功定时器

4.1.13.3 停止并清除软件定时器

功能	停止并清除软件定时器
函数定义	void drv_timer_stop(drv_timer_t *timer);
参数常量	Time=软定时器
返回	无

4.1.13.4 释放所有软件定时器

功能	释放软件定时器
函数定义	void drv_timer_free();
参数常量	无
返回	无

4.1.13.5 定时器中断服务接口

功能	DRV 软件定时服务，通过定时中断服务调用

函数定义	void drv_timer_service(u32_t arg)
参数常量	Arg
返回	无

4.1.13.6 获取程序计数器计数

功能	获取程序计数器滴答计数，1ms 增加一个计数单位
函数定义	u32_t drv_timer_tick_get(void) drv_timer_tick_in_isr_get()
参数常量	无
返回	返回 32 位计数

4.1.13.7 打印软件使能的计时器

功能	打印软件使能的计时器
函数定义	void drv_timer_dump()
参数常量	无
返回	无

4.1.13.8 获取定时器启动标志状态

功能	检查定时器是否已启动
函数定义	drv_timer_started(ptr)
参数常量	Ptr=定时器
返回	1 启动或者 0 未启动

4.1.13.9 获取定时器停止标志位状态

功能	检查定时器是否已停止
函数定义	drv_timer_stopped(ptr)
参数常量	Ptr=定时器
返回	1 停止或者 0 运行

4.1.13.10 检查定时器溢出状态

功能	检查定时器是否已经溢出
函数定义	drv_timer_expired(ptr)
参数常量	Ptr=定时器
返回	1 溢出或者 0 未溢出

4.1.14 UART 驱动

Uart 函数文件 Drv_uart.h 和 Drv_uart.c

4.1.14.1 UART 初始化

功能	UART 初始化,设置控制流，停止位，奇偶校验，波特率，uart 中断使能
函数定义	void drv_uart_common_init(DRV_UART_ID_t uart_id);
参数常量	Id=uart 口，0 或者 1
返回	无

4.1.14.2 设置默认的 uart 端口

功能	设置默认的 uart 端口
函数定义	void drv_uart_default_set(DRV_UART_ID_t uart_id)
参数常量	Id=uart 口，0 或者 1
返回	无

4.1.14.3 获取默认的第一个 uart 口

功能	获取默认的第一个 uart 口
函数定义	drv_uart_default_get()
参数常量	无
返回	返回 0 后者 1

4.1.14.4 获取第二个 uart 口

功能	获取第二个 uart 口
函数定义	drv_uart_second_get()
参数常量	无
返回	返回 0 后者 1

4.1.14.5 配置 uart 波特率

功能	配置 uart 波特率
函数定义	void drv_uart_common_baud_rate_set(DRV_UART_ID_t uart_id, u32_t baudrate);
参数常量	Uart_ID=UART 口 0 或者 1；baudrate=待设置的波特率值
返回	无

4.1.14.6 获取 uart 的当前波特率

功能	获取 uart 的波特率
函数定义	u32_t drv_uart_common_baud_rate_get(DRV_UART_ID_t uart_id);
参数常量	Uart_id=uart 口 0 或者 1
返回	返回当前 uart 的波特率的值

4.1.14.7 UART 读取字节

功能	UART 读取一个字节
函数定义	u8_t drv_uart_common_getbyte(DRV_UART_ID_t uart_id, u8_t *c);
参数常量	Uart_id=uart 口，0 或者 1；*c=读取的一个字节的数据
返回	读取成功返回一个字节，没有成功返回 0

4.1.14.8 UART 等待读取一个字节数据

功能	UART 等待读取一个字节的数据
函数定义	u8_t drv_uart_common_waitbyte(DRV_UART_ID_t uart_id);
参数常量	Uart_id=0 或者 1
返回	返回一个字节的数据

4.1.14.9 UART 发送一个字节数据

功能	UART 写入一个字节的数据
函数定义	void drv_uart_common_putbyte(DRV_UART_ID_t uart_id, u8_t c);
参数常量	Uart_id=0 或者 1；c=8bit 数据
返回	无

4.1.14.10 UART 发送单个字符

功能	UART 写入单个字符
函数定义	int drv_uart_common_putchar (DRV_UART_ID_t uart_id, char c)
参数常量	Uart_id=0 或者 1；c=字符
返回	1

4.1.14.11 UART 发送字符串

功能	UART 发送字符串
函数定义	void drv_uart_common_putstr(DRV_UART_ID_t uart_id, const char *ptr) void drv_uart_common_puts(DRV_UART_ID_t uart_id, const char *ptr)
参数常量	Uart_id=0 或者 1；ptr=发送的字符串，指针类型
返回	无

4.1.14.12 uart 发送一个整数

功能	uart 发送一个整数
函数定义	int putchar(int c)
参数常量	C=发送的整数
返回	1

4.1.14.13 uart 以 10 进制格式发送数据

功能	Uart 将数据以 10 进制的格式发送并输出
函数定义	void drv_uart_common_putdec(DRV_UART_ID_t uart_id, u32_t v) drv_uart_putdec(v) drv_uart1_putdec(v)
参数常量	Uart_id=0 或者 1；v=32bit 数据
返回	无

4.1.14.14 uart 输出双精度数据

功能	uart 输出双精度数据
函数定义	void drv_uart_common_putdouble(DRV_UART_ID_t uart_id, double v) drv_uart_putdouble(v) drv_uart1_putdouble(v)
参数常量	Uart_id=0 或者 1；v=double 类型数据
返回	无

4.1.14.15 uart 输出指定长度的 16 进制数据

功能	输出 len 长度的指定数据，可以输出地址，并且以 16 进制输出
函数定义	void drv_uart_common_puthex(DRV_UART_ID_t uart_id, u8_t *pval, int len, bool_t with_addr) drv_uart_puthex(pval,len,with_addr) drv_uart1_puthex(pval,len,with_addr)
参数常量	Uart_id=0 或者 1；pval=发送数据；len=数据长度；with_addr=0 或者 1，标志位，1 输出地址
返回	无

4.1.14.16 uart1 dmn 接收中断处理

功能	uart1 dmn 接收中断处理
函数定义	static void drv_uart1_dma_handler(u32_t arg)
参数常量	Arg;
返回	无

4.1.14.17 uart1 启用 DMA 接收

功能	uart1 启用 DMA 接收
函数定义	void drv_uart1_dma_en_set(bool_t en)
参数常量	En=1 或者 0,1 使能，0 关闭
返回	无

4.1.14.18 uart0 初始化以及相关功能函数接口

功能	Uart0 初始化，接收发送功能接口，详细请参照 4.1.14.7~4.1.14.11
函数定义	drv_uart_init() drv_uart_getbyte(pchar) drv_uart_waitbyte() drv_uart_putbyte(chr) drv_uart_putstr(ptr) drv_uart_putchar(chr) drv_uart_puts(ptr)
参数常量	
返回	

4.1.14.19 uart1 初始化以及相关功能函数接口

功能	Uart0 初始化，接收发送功能接口，详细请参照 4.1.14.7~4.1.14.11
函数定义	drv_uart1_init() drv_uart1_getbyte(pchar) drv_uart1_waitbyte() drv_uart1_putbyte(chr) drv_uart1_putstr(ptr) drv_uart1_putchar(chr) drv_uart1_puts(ptr)
参数常量	
返回	

4.1.14.20 Uart0 输出 16 进制数据

功能	Uart0 输出 16 进制数据
函数定义	drv_uart_puthex64(val) drv_uart_puthex32(val) drv_uart_puthex16(val) drv_uart_puthex8(val) drv_uart_puthex4(val)
参数常量	Val=传输数据
返回	无

4.1.14.21 uart1 输出 16 进制数据

功能	uart1 输出 16 进制数据
函数定义	drv_uart1_puthex64(val) drv_uart1_puthex32(val) drv_uart1_puthex16(val) drv_uart1_puthex8(val) drv_uart1_puthex4(val)
参数常量	Val=传输数据
返回	无

4.1.14.22 uart 配置流量控制

功能	uart 配置流量控制
函数定义	drv_uart_common_flowctrl_set(uart_id,flg)
参数常量	uart_id=0 或者 1，uart 端口；flg=0 或者 1，1 打开，0 关闭
返回	无

4.1.14.23 获取 uart 流量控制状态

功能	获取 uart 流量控制状态
函数定义	drv_uart_common_flowctrl_get(uart_id)
参数常量	uart_id=0 或者 1，uart 端口
返回	1 打开；0 关闭

4.1.14.24 uart 设置数据长度

功能	uart 设置数据长度
函数定义	drv_uart_common_data_length_set(uart_id,len)
参数常量	Id=0 或者 1；len=数据长度（0~3），0 代表 5 位，1 代表 6 位，2 代表 7 位，3 代表 8 位
返回	无

4.1.14.25 设置 uart 奇偶校验位

功能	设置 uart 奇偶校验位
函数定义	drv_uart_common_parity_check_set(uart_id,en,even)
参数常量	id=uart 端口，1 或者 0；en=使能标志位，1 打开奇偶校验，0 关闭奇偶校验；even=奇偶校验选择，0 选择奇数校验，1 选择偶数校验
返回	无

4.1.14.26 获取 uart 奇偶校验使能状态

功能	获取 uart 奇偶校验使能状态
函数定义	drv_uart_common_parity_check_get(uart_id)
参数常量	id=uart 端口，1 或者 0
返回	0 关闭或者 1 打开校验

4.1.14.27 获取奇偶校验模式选择

功能	获取奇偶校验模式选择
函数定义	drv_uart_common_parity_selection_get(uart_id)
参数常量	id=uart 端口，1 或者 0
返回	0 奇数校验或者 1 偶数校验

4.1.14.28 设置 uart 停止位

功能	设置 uart 停止位
函数定义	drv_uart_common_stop_bit_set(uart_id,two_bits)
参数常量	id=uart 端口，1 或者 0；two_bit=停止位设置，0 代表 1 位停止位，1 代表 2 位停止位
返回	无

4.1.14.29 获取 uart 停止位设置

功能	获取停止位的位数
函数定义	drv_uart_common_stop_bit_get(uart_id)
参数常量	id=uart 端口，1 或者 0
返回	0 或者 1,0 代表 1 位停止位，1 代表 2 位停止位

4.1.15 看门狗定时器定义

函数文件 Drv_wdt.c 或者 Drv_wdt.h

4.1.15.1 生成服务序列以重新加载定时器

功能	喂狗
函数定义	void drv_wdt_feed(void);

参数常量	无
返回	无

4.1.15.2 配置看门狗定时器并启动

功能	配置看门狗定时器并启动
函数定义	void drv_wdt_init(u32_t timeout, bool_t enable);
参数常量	Timeout=看门狗超时值；enable=使能参数，1 启用看门狗，0 关闭看门狗
返回	无

4.1.15.3 获取看门狗超时值

功能	获取看门狗超时值
函数定义	u32_t drv_wdt_timeout_get(void);
参数常量	无
返回	返回 32 位超时值

4.1.15.4 获取看门狗的使能状态

功能	获取看门狗使能状态
函数定义	drv_wdt_enable_get()
参数常量	无
返回	返回 0 或者 1，0 关闭，1 使能

4.2 系统适配层的 C 库

4.2.1 ctype 模式检查

函数文件 Ctype.c，Ctype.h

4.2.1.1 将 16 进制字符转换成 10 进制

功能	检查 ctype
函数定义	unsigned char toxdigit(char c);
参数常量	c=输入
返回	是 16 进制字符输出 0~15，不是 255

4.2.1.2 检查 x 是否是 ctype 表中的掩码

功能	检查 x 是否在 g_ctye_tb 表中

函数定义	__ismask(x)
参数常量	X=输入字符
返回	在表中返回 0，1,2,4,8,16,32,64，128，40,65,66，160

4.2.1.3 检查 c 是否为数字字母

功能	检查 c 是否为数字字母
函数定义	isalnum(c)
参数常量	C=输入字符
返回	0 或者 1,0 不是字母或者数字，1 是字母或者数字

4.2.1.4 检查 c 是否为字母

功能	检查 c 是否为字母
函数定义	isalpha(c)
参数常量	C=输入字符或者数字
返回	0 或者 1,0 不是字母，1 是字母

4.2.1.5 检查 c 是否是控制字符

功能	检查 c 是不是控制字符
函数定义	iscntrl(c)
参数常量	C=输入字符
返回	0 或者 1,0 不是控制，1 是控制字符

4.2.1.6 检查 c 是否可打印字符

功能	检查 c 是否可打印字符，不包括空格
函数定义	isgraph(c)
参数常量	C=输入字符
返回	0 或者 1,0 不是可打印字符，1 是可打印字符

4.2.1.7 检查 c 是不是小写字母

功能	检查 c 是不是小写字母（a~z）
函数定义	islower(c)
参数常量	C=输入字符
返回	0 或者 1,0 不是小字母，1 是小写字母

4.2.1.8 检查 c 是否可打印字符

功能	检查 c 是否可打印字符

函数定义	isprint(c)
参数常量	C=字符输入
返回	0 或者 1,0 不可打印，1 可打印

4.2.1.9 检查 c 是不是标点符号

功能	检查输入 c 是不是标点符号
函数定义	4.2.1.2(c)
参数常量	C=字符输入
返回	0 或者 1,0 不是，1 是

4.2.1.10 检查 c 是不是空格字符函数

功能	检查 c 是不是空格字符
函数定义	isspace(c)
参数常量	c=字符输入
返回	0 或者 1,0 不是，1 是

4.2.1.11 检查字符是不是大写字符

功能	检查字符是不是大写字符
函数定义	isupper(c)
参数常量	c=字符输入
返回	0 或者 1,0 不是，1 是

4.2.1.12 检查 c 是不是数字

功能	检查输入字符是不是数字
函数定义	isdigit(c)
参数常量	c=输入字符
返回	0 或者 1,0 不是，1 是

4.2.1.13 检查 c 是不是 16 进制字符

功能	检查输入字符是不是 16 进制字符
函数定义	isxdigit(c)
参数常量	C=字符输入
返回	0 或者 1,0 不是，1 是

4.2.1.14 检查 c 是不是 ascii 码字符

功能	检查 c 是不是 ascii 码字符

函数定义	isascii(c)
参数常量	C=输入
返回	0 或者 1,0 不是，1 是

4.2.1.15 将 c 转换成 ascii 码字符

功能	将 c 转换成 ascii 码字符
函数定义	toascii(c)
参数常量	C=输入
返回	Ascii 字符

4.2.1.16 检查 c 为小写字符并转换成大写字符

功能	检查 c 为小写字符并转换成大写字符
函数定义	toupper(c)
参数常量	C=输入
返回	小写字符，或者 c 本身

4.2.1.17 检查 c 是否是大写字符并转换成小写字母

功能	检查 c 是否是小写字符并转换成小写字母
函数定义	tolower(c)
参数常量	C=输入
返回	大写字母，或者 c 本身

4.2.2 消息队列

消息队列功能仅在非操作系统模式下有效！

4.2.2.1 消息队列初始化

功能	消息队列初始化
函数定义	void msq_init(void);
参数常量	无
返回	无

4.2.2.2 推送消息队列数据入栈

功能	推送消息队列数据入栈
函数定义	void msq_push(msq_t msg);

参数常量	Msg=入栈的消息
返回	无

4.2.2.3 消息数据推送到队列

功能	消息数据推送到队列
函数定义	void msq_push_msg(u32_t msq_type, u32_t val1, u32_t val2, u32_t val3);
参数常量	msq_type=消息类型,（0-空消息,1-uart0 消息,2-uart1 消息,3-i2c0 从机模式消息,4-i2c1 从机模式消息,5-spi 从机模式消息,6-扩展中断消息,15-测试消息）val1=数据 1；val2=数据 2；val3=数据 3
返回	无

4.2.2.4 弹出消息队列数据

功能	弹出消息队列数据
函数定义	void msq_pop(msq_t *p_msg);
参数常量	Msg=弹出得消息，指针类型
返回	无

4.2.2.5 获取传入消息的类型

功能	获取传入消息的类型
函数定义	SAL_MSQ_TYPE(msgptr)
参数常量	Msgptr=消息类型指针
返回	0-空消息,1-uart0 消息,2-uart1 消息,3-i2c0 从机模式消息,4-i2c1 从机模式消息,5-spi 从机模式消息,6-扩展中断消息,15-测试消息

4.2.2.6 获取传入消息的第一个数据

功能	获取传入消息的第一个数据
函数定义	SAL_MSQ_VALUE1(msgptr)
参数常量	Msgptr=消息类型指针
返回	8 位的数据

4.2.2.7 获取传入消息的第二个数据

功能	获取传入消息的第二个数据
函数定义	SAL_MSQ_VALUE2(msgptr)
参数常量	Msgptr=消息类型指针
返回	返回 8 位的数据

4.2.2.8 获取传入消息的第三个数据

功能	获取传入消息的第三个数据
函数定义	SAL_MSQ_VALUE3(msgptr)
参数常量	Msgptr=消息类型指针
返回	8 位数据

4.2.3 字符输出函数库

4.2.3.1 支持长度的格式化输出字符串到指定缓冲区（va_list）

功能	将可变参数格式化输出到一个 buf 里
函数定义	int vsnprintf(char *buf, int bsize, const char *fmt, va_list ap);
参数常量	buf=存放生产的格式化字符串；bsize=可输出字符串限制长度；fmt=指定输出格式的字符串；ap=指可变参数
返回	生产成功返回字符串的长度，长度超过 size 返回，原字符串的长度

4.2.3.2 支持长度的格式化输出字符串到指定缓冲区

功能	将可变参数”...”按照 fmt 格式，输入到 buf 里
函数定义	int snprintf(char *buf, int bsize, const char *fmt, ...);
参数常量	buf=目标字符串；bsize=拷贝的限制长度；fmt=源字符串
返回	返回字符串的长度

4.2.3.3 格式化输出字符串到指定缓冲区

功能	将格式化字符换输出到字符串里
函数定义	int sprintf(char *buf, const char *fmt, ...);
参数常量	Buf=目标字符串指针；fmt=格式化输出字符串
返回	字符串的长度

4.2.3.4 格式化输出字符串到标准输出接口

功能	将格式化字符串输出到标准输出
函数定义	int printf(const char *fmt, ...);
参数常量	fmt=格式化字符串
返回	输出字符串的长度

4.2.3.5 串口输出字符串

功能	串口输出字符串
函数定义	int puts(const char *str)
参数常量	Str=字符串地址
返回	换行符

4.2.4 标准库

4.2.4.1 绝对值

功能	获取整形数的绝对值
函数定义	int abs(int);
参数常量	int=整形数据
返回	无符号整数

4.2.4.2 字符串转整形

功能	字符串转整形
函数定义	int strtoi(const char *p);
参数常量	P=字符串指针
返回	返回整数

4.2.4.3 字符串转换为无符号长整形

功能	10 进制，8 进制，16 进制，2 进制样式字符串转换为无符号长整形
函数定义	unsigned long strtoul(const char *s, char **end, int base)
参数常量	S=字符串指针；end=二级指针，用于存放剩余字符串；base=基准
返回	无符号长整形

4.2.4.4 字符串转换长整形

功能	字符串转换长整形
函数定义	long strtol(const char *nptr, char **endptr, int base)
参数常量	nptr=字符串指针；endptr=二级指针，存放剩余字符串；base=基准
返回	长整形

4.2.4.5 随机数发生器的初始化函数

功能	随机数发生器的初始化函数
函数定义	void srand(unsigned int seed);
参数常量	Seed=随机输入
返回	无

4.2.4.6 伪随机数发生器

功能	伪随机数发生器
函数定义	int rand(void);
参数常量	无
返回	返回一个随机整数

4.2.4.7 无符号 32 位伪随机整数发生器

功能	32 位无符号伪随机数发生器
函数定义	int rand(void);
参数常量	无
返回	返回一个无符号 32 位随机整数

4.2.5 字符串系统适配层

4.2.5.1 16 进制字符串转换为数字数组

功能	16 进制字符串转换为数字数组
函数定义	int strtohex(unsigned char *dst, char *src, int len);
参数常量	dst 目标数字数组；src 16 进制字符串；len 指数据长度
返回	4.2.5.1 **返回数据的长度**

4.2.5.2 字符串比较 n 个字节

功能	字符串比较
函数定义	int memcmp(const void *str1, const void *str2, register int n)
参数常量	str1=字符串 1；str2=字符串 2；n=长度
返回	s1>s2 返回>0;s1<s2 返回<0;s1=s2,返回 0

4.2.5.3 字符串比较

功能	字符串比较
函数定义	int strcmp(const char *s1, const char *s2)
参数常量	s1=字符串 1；s2=字符串 2
返回	s1>s2; >0 s1==s2; 0 s1<s2; <0

4.2.5.4 字符串比较最多 n 个字节

功能	字符串比较
函数定义	int strncmp(const char *s1, const char *s2, register int n)
参数常量	str1=字符串 1；str2=字符串 2；n=长度
返回	s1>s2; >0 s1==s2; 0 s1<s2; <0

4.2.5.5 不区分大小写比较字符串前 n 个字节

功能	**不区分大小写比较字符串前 n 个字节**
函数定义	int strncasecmp(const char *s1, const char *s2, register int n)
参数常量	str1=字符串 1；str2=字符串 2；n=长度
返回	s1>s2: >0 s1==s2: 0 s1<s2: <0

4.2.5.6 不区分大小写比较字符串

功能	不区分大小写比较字符串
函数定义	int strcasecmp(const char *s1, const char * s2)
参数常量	str1=字符串 1；str2=字符串 2
返回	s1>s2: >0 s1==s2: 0 s1<s2: <0

4.2.5.7 内存数据复制

功能	将源地址数据复制指定长度到目标地址空间
函数定义	void *memcpy2 (void * dst0,const void * src0,unsigned long len0)
参数常量	dst0=目标地址；src0=源数据地址；len0=复制长度
返回	返回指向目标数据的指针

4.2.5.8 内存数据移动

功能	从源地址移动指定长度到目标地址
函数定义	void *memmove2 (void *dst_void,const void *src_void,unsigned long length)
参数常量	dst0=目标地址空间；src0=源地址空间；len0=复制长度
返回	指向目标数据的指针

4.2.5.9 内存初始化函数

功能	将一块内存初始化为指定的值，用于数组和结构体的快速初始化
函数定义	void *memset(register void *str, register int c, register int n)
参数常量	Str=目标内存指针；c=初始化的值；n=初始化空间的大小
返回	指向内存的指针

4.2.5.10 字符查找

功能	在一个字符串中查找首次出现指定字符
函数定义	char *strchr(register char *sp, register char c)
参数常量	Sp=目标字符串；c=需要查找的字符
返回	返回 sp 中出现的 c 字符位置；没有找到返回 0

4.2.5.11 获取字符串的长度

功能	获取字符串的长度
函数定义	int strlen(register char *s)
参数常量	S=目标字符串
返回	字符串的长度

4.2.5.12 获取字符串长度并限制最大长度

功能	获取字符串长度并限制最大长度
函数定义	int strnlen(const char *s, unsigned int maxlen)
参数常量	s=目标字符串；maxlen=限制长度
返回	字符串长度，最大 maxlen

4.2.5.13 判断是不是字符串子串

功能	判断 find 是不是 s1 的字串
函数定义	char *strstr(char *s1, char *find)

参数常量	s1=指向源字符串；find=查找字符串地址
返回	查找到返回处于 s1 位置指针，没有查找到返回 0

4.2.5.14 字符串复制

功能	原字符串复制到目标字符串
函数定义	char *strcpy(register char *s1, const char *s2)
参数常量	s1=目标字符串；s2=源字符串
返回	返回指向目标字符串的指针

4.2.5.15 字符串连接

功能	两个字符串连接，最多连接 n 个
函数定义	char *strncat(register char *s1,register char *s2,register int n)
参数常量	S1=被连接的字符串；s2=连接的字符串；n=连接限制长度
返回	返回指向连接后的字符串指针

4.2.5.16 带限定长度的字符串复制

功能	复制 n 个字节字符到指定字符串
函数定义	char *strncpy(register char *s1, register char *s2, register int n)
参数常量	S1=目标字符串；s2=源字符串；n=限定长度
返回	指向目标字符串指针

4.2.5.17 计算字符串 string 有几个连续的字符属于字符串 charset

功能	计算 string 有几个连续的字符属于 charset
函数定义	int strspn(char *string, register char *charset)
参数常量	string=字符串 1；charset=字符串 2
返回	返回字符串 string 开头连续包含字符串 charset 内的字符数目

4.2.5.18 检索目标字符串中第一个匹配到源字符串的字符

功能	检索目标字符串中第一个匹配到源字符串的字符
函数定义	char *strpbrk(register char *string,register char *brkset)
参数常量	string=目标字符串；brkset=源字符串
返回	string 中出现的第一个字符的位置

4.2.5.19 字符查找并返回最右出现位置

功能	字符查找并返回最右出现位置
函数定义	char *strrchr(register char *sp, register char c)
参数常量	sp=目标字符串；c=查找的字符

返回	返回 sp 最右出现该字符的位置

4.2.5.20 字符串分割

功能	将字符串 string 以 sepset 字符分割
函数定义	char *strtok(char *string, char *sepset)
参数常量	string=要分解的字符串；sepset=分割字符
返回	被分割后的字符串 string

4.2.5.21 字符串中大写字母转换为小写字母

功能	字符串中大写字母转换为小写字母
函数定义	char * strtolower(char *s)
参数常量	s=目标字符串
返回	转变后的字符串

4.2.5.22 字符串中小写字母转换为大写字母

功能	字符串中小写字母转换为大写字母
函数定义	char *strtoupper(char *s)
参数常量	s=目标字符串
返回	转变后的字符串

4.2.6 软件定时器

函数文件 time.c

4.2.6.1 设置并启动定时器

功能	设置并启动定时器
函数定义	void timer_start(u32_t interval, volatile timer_t *timer);
参数常量	Interval=定时器间隔；*timer=定时器
返回	无

4.2.6.2 检查定时器是否溢出

功能	检查定时器是否已过期
函数定义	int timer_expired(volatile timer_t *timer);
参数常量	Time=定时器
返回	1 溢出或者 0 没有溢出

4.2.7 系统适配层实用程序

函数文件 Utils.c 和 Utils.h

4.2.7.1 延迟

功能	延迟
函数定义	void delay(int cnt);
参数常量	cnt=延迟时间
返回	无

4.2.7.2 设置位数获取

功能	设置位数获取
函数定义	u32_t set_bits_get(u32_t value);
参数常量	value=数据
返回	返回置位的数量

4.2.7.3 清 0 位数获取

功能	清零位数获取
函数定义	u32_t clear_bits_get(u32_t value);
参数常量	value=数据
返回	返回清零的数量

4.2.7.4 检查数据的某位的是否设置

功能	检查数据的某位的是不是被设置
函数定义	SAL_IS_BIT_SET(val,pos)
参数常量	val=数据；pos=位置
返回	0 没有设置或者 1 已设置

4.2.7.5 检查数据某位是否被清 0

功能	检查数据某位是否被清 0
函数定义	SAL_IS_BIT_CLEARED(val,pos)
参数常量	val=数据
返回	1 清除或者 0 未清除

4.2.7.6 数据某位清 0

功能	清 0 数据的指定位
函数定义	SAL_BIT_CLEAR(val,pos)
参数常量	val=数据；pos=位置
返回	pos 位清 0 的数据

4.2.7.7 数据某位设置为 1

功能	数据指定位设置为 1
函数定义	SAL_BIT_SET(val,pos)
参数常量	val 数据；pos 位置
返回	处理后的数据

4.2.7.8 获取数据某段的值

功能	获取数据某段的值
函数定义	SAL_FIELD_GET(dst,reg,field)
参数常量	dst 数据；reg 寄存器宏定义；field 宏定义
返回	数据的某段的值

4.2.7.9 设置数据某段的值

功能	设置数据某段的值位 val
函数定义	SAL_FIELD_SET(dst,reg,field,val)
参数常量	Dst=数据；reg 寄存器宏定义；field 位置宏定义；val 需要设置的数据
返回	设置好的数据

4.2.7.10 清除数据的字段

功能	清除数据某段的值
函数定义	SAL_FIELD_CLEAR(dst,reg,field)
参数常量	dst 数据；reg 寄存器宏定义；field 位置宏定义
返回	清除字段的数据

4.2.7.11 设置数据某宽度字段的值

功能	设置数据某宽度字段的值
函数定义	SAL_FIELD_SET2(dst,field,width,val)
参数常量	dst 数据；field 位置；width 宽度；val 要设置的数据
返回	设置后的数据

4.2.7.12 分别获取 32 位数据 0~7，8~15,16~23,24~31 位的数据

功能	分别获取 32 位数据 0~7，8~15,16~23,24~31 位的数据
函数定义	SAL_U32_BYTE0(val)；SAL_U32_BYTE1(val) SAL_U32_BYTE2(val)；SAL_U32_BYTE3(val)
参数常量	val32 位数据
返回	无符号 8 位数据

4.2.7.13 8 位数据转换成 32 位数据

功能	将 4 个 8 位数据转换成 32 位数据
函数定义	SAL_4BYTES_U32(val1,val2,val3,val4)
参数常量	val1 低 8 位数据；val2（8~15 位数据）；val3（16~23 位数据）；val4 高 8 位数据
返回	32 位无符号数据

4.2.7.14 获取无符号 16 位数据高低 8 位数据

功能	获取无符号 16 位数据高低 8 位数据
函数定义	SAL_U16_BYTE0(val)；SAL_U16_BYTE1(val)
参数常量	val 16 位无符号数据
返回	无符号 8 位数据

4.2.7.15 合成无符号 16 位数据

功能	合成无符号 16 位数据
函数定义	SAL_2BYTES_U16(val1,val2)
参数常量	val1 高 8 位数据；val2 低 8 位数据
返回	无符号 16 位数据

4.2.7.16 大小端数据获取高 8 位

功能	大小端数据获取高 8 位
函数定义	SAL_U16_MSB_BYTE(val)；SAL_U16_LSB_BYTE(val)
参数常量	val 16 位数据
返回	无符号 8 位数据

4.2.7.17 数据高低位互换

功能	数据高低位互换
函数定义	SWAP_2BYTES(val)；SWAP_4BYTES(val)
参数常量	val 16 位数据

返回	无符号 8 位数据

4.2.7.18 数据 x 转换位 u32_t/u16_t 的整数倍

功能	数据 x 缩小转换位 u32_t/u16_t 的整数倍，多余的部分去掉
函数定义	UNALIGNED_32BITS(X) UNALIGNED_16BITS(X)
参数常量	x 数据
返回	转换后的数据

4.2.7.19 内存对齐

功能	将 x 扩展转换为 u32_t 的整数倍，多余的位置用 0 替代
函数定义	ALIGN_32BITS(X)
参数常量	X 数据
返回	转换后的数

4.2.8 XModem 接口

4.2.8.1 XModem 接收处理函数

功能	XModem 接收传输数据
函数定义	long xmodem_rx(unsigned long dest, unsigned long destsz);
参数常量	Dest 写入的地址；destsz 写入的数据长度
返回	接收成功返回数据的长度，失败返回负数

4.2.8.2 接收数据校验

功能	校验
函数定义	static int check(int crc, const unsigned char *buf, int sz)
参数常量	crc 标志位，为 1 支持 crc 校验；buf 接收的数据；sz 接收数据的长度
返回	校验成功返回 1，失败返回 0

4.2.8.3 限定时间内获取 uart 数据

功能	限定时间内获取 uart 数据
函数定义	int port_inbyte(unsigned int tn)
参数常量	tn 时间等级=1 为 1s
返回	成功返回读取的值，失败返回 0

4.2.8.4 向 uart 接口写入一个数据

功能	向 uart 接口写入一个数据
函数定义	void port_putbyte(int c)
参数常量	c 写入的数据
返回	无

4.2.9 CCITT CRC16

4.2.9.1 获取 CCITT CRC16 校验码

功能	获取 CRC 十六位校验码
函数定义	unsigned short crc16_ccitt(const unsigned char *buf, int len);
参数常量	buf 数据常量指针，len 指数据长度
返回	16 位 crc

5 PD 库

Pd 库包括 pd 配置，设备策略管理，type-c 连接配置，协议层消息收发处理，qc 快充模式兼容，pe 策略引擎等主要模块

5.1 PD 配置

5.1.1 pd 参数获取以及基本配置

Pd 的 gp_usbpd_cfg 一般都会从 flash 里面读取配置，函数接口

5.1.1.1 flash 获取配置数据

功能	从 flash 里获取系统参数
函数定义	void sys_cfg_load(void)
参数常量	无
返回	无

5.1.1.2 初始化 gp_usbpd_cfg

功能	初始化 gp_usbpd_cfg 结构体，log 等级以及设置 log 等级回调函数，系统设置保存回调函数
函数定义	void sys_pd_cfg_load(void)
参数常量	无
返回	无

5.1.1.3 pd 初始化配置参数

功能	Pd 初始配置函数
函数定义	sys_pd_cfg_init()
参数常量	无
返回	无

5.1.1.4pd-src 改变固定电源功率标志位清除 0

功能	清除 Pd -src 改变供电功率设置标志位
函数定义	void sys_cfg_onsaving(void)
参数常量	无
返回	无

5.1.1.5 设置系统 debug 调试等级

功能	设置系统 debug 调试等级

函数定义	void sys_cfg_dbg_onchange(u32_t level)
参数常量	Level=0~5
返回	无

5.1.1.6 pd 初始化

功能	Pd 初始化
函数定义	void usbpd_init(void);
参数常量	无
返回	无

5.1.1.7 pd 中断处理函数

功能	Pd 中断处理
函数定义	void usbpd_isr(u32_t int_status)
参数常量	Int_status, 中断状态
返回	无

5.1.1.8 usbpd 主函数

功能	Usbpd 主函数
函数定义	void usbpd_main()
参数常量	无
返回	无

5.1.2 sink 模式配置

```
gp_usbpd_cfg->port_cfg.pwr_src = POWER_SOURCE_BATTERY;
    电源供给类型，sink 一般都是电池
gp_usbpd_cfg->port_cfg.def_pdr = PD_ROLE_CONSUMER;
    Pd 角色，sink 为接收者
gp_usbpd_cfg->port_cfg.def_usb_pr = POWER_ROLE_SINK;
    端口电源角色
gp_usbpd_cfg->port_cfg.def_usb_dr = USB_DATA_ROLE_UFP;
    Usb 数据角色
 gp_usbpd_cfg->port_cfg.try_src = FALSE;
    连接时首先尝试作为 source，关闭
gp_usbpd_cfg->port_cfg.try_snk = FALSE;
    连接时首先尝试作为 sink，可以打开，可以关闭
 gp_usbpd_cfg->port_cfg.dual_role_pwr = FALSE;
    是不是 drp，关闭
 gp_usbpd_cfg->port_cfg.usb_susp_supp = FALSE;
    Usb 休眠，不支持关闭
 gp_usbpd_cfg->port_cfg.ext_pwr = FALSE;
    是不是外部供电，关闭
 gp_usbpd_cfg->port_cfg.usb_comm_cap = FALSE;
```

```
    Usb 通讯能里设置
gp_usbpd_cfg->port_cfg.drs_cap = FALSE;
    数据角色交换能力，不支持
gp_usbpd_cfg->port_cfg.vdm_en = VDM_MODE_OFF;
    供应商定义消息模式，默认关闭
```

5.1.3 Source 模式配置

```
gp_usbpd_cfg->port_cfg.pwr_src = POWER_SOURCE_AC_INPUT;
    电源供给类型，src 一般都为交流电输入类型
gp_usbpd_cfg->port_cfg.def_pdr = PD_ROLE_PROVIDER;
    Pd 角色，src 供给者
gp_usbpd_cfg->port_cfg.def_usb_pr = POWER_ROLE_SOURCE;
    端口电源角色
gp_usbpd_cfg->port_cfg.def_usb_dr = USB_DATA_ROLE_DFP;
    Usb 数据角色
gp_usbpd_cfg->port_cfg.try_src = FALSE;
    首次连接尝试作为 src
gp_usbpd_cfg->port_cfg.try_snk = FALSE;
    首次连接尝试作为 snk，关闭
gp_usbpd_cfg->port_cfg.dual_role_pwr = FALSE;
    drp 角色，关闭
gp_usbpd_cfg->port_cfg.usb_susp_supp = FALSE;
    Usb 休眠能力，默认关闭
gp_usbpd_cfg->port_cfg.ext_pwr = TRUE;
    外部供电，打开
gp_usbpd_cfg->port_cfg.usb_comm_cap = FALSE;
    Usb 通信能力
gp_usbpd_cfg->port_cfg.drs_cap = FALSE;
    数据角色交换能力，默认不支持
gp_usbpd_cfg->port_cfg.vdm_en = VDM_MODE_OFF;
    供应商自定义消息模式，默认关闭
```

5.1.4 drp 模式配置

```
gp_usbpd_cfg->port_cfg.pwr_src ;
    电源供给类型，drp 可以作为 src，也可以作为 snk，
    此处可以选择 POWER_SOURCE_AC_INPUT，也可以选择
    POWER_SOURCE_BATTERY
gp_usbpd_cfg->port_cfg.def_pdr ;
    Pd 角色也有两种，供给者或者接收者
gp_usbpd_cfg->port_cfg.def_usb_pr = POWER_ROLE_DRP;
    端口电源角色
gp_usbpd_cfg->port_cfg.def_usb_dr = USB_DATA_ROLE_DRD;
    默认的 usb 数据角色
gp_usbpd_cfg->port_cfg.try_src = FALSE;
    首次连接尝试作为 src
gp_usbpd_cfg->port_cfg.try_snk = FALSE;
    首次连接尝试作为 snk
gp_usbpd_cfg->port_cfg.dual_role_pwr = TRUE;
    数据角色交换能力，支持
```

gp_usbpd_cfg->port_cfg.usb_susp_supp = FALSE;
Usb 休眠能力，默认不支持
gp_usbpd_cfg->port_cfg.ext_pwr = TRUE;
外部电源供给，打开
gp_usbpd_cfg->port_cfg.usb_comm_cap = FALSE;
Usb 通讯能力，关闭
gp_usbpd_cfg->port_cfg.drs_cap = TRUE;
Drp 能力，支持
gp_usbpd_cfg->port_cfg.vdm_en = VDM_MODE_OFF;
供应商自定义消息，默认关闭

5.1.5 插头电缆模式配置

gp_usbpd_cfg->cable_plug = TRUE;
标志位使能，打开 cable plug 模式
gp_usbpd_cfg->capture_mode = FALSE;
关闭捕获模式
gp_usbpd_cfg->prod_type = UPD_PRODUCT_ACTIVE_CABLE;
默认配置有源电缆设备
gp_usbpd_cfg->port_cfg.def_pdr = PD_ROLE_CABLE_PLUG
Pd 的角色，设置为 PD_ROLE_CABLE_PLUG

5.1.6 捕获模式

gp_usbpd_cfg->capture_mode = TRUE;
使能开关，打开 capture mode
gp_usbpd_cfg->cable_plug = FALSE;
关闭电缆插头模式
gp_usbpd_cfg->prod_type = UPD_PRODUCT_UNDEFINED;
连接设备类型，捕获模式未定义

5.1.7 Qc 模式

gp_usbpd_cfg->port_cfg.quick_charge_en
快充使能标志位，1 使能 0 关闭
gp_usbpd_cfg->port_cfg.quick_charge_ver = DPM_QC_V3;
Qc 协议版本设置

5.1.8 基本信息配置

gp_usbpd_cfg->port_num = 1;
端口配置，只读
gp_usbpd_cfg->vd_usb.svid = PD_STANDARD_VID;
Pd 标准 vid
gp_usbpd_cfg->fw_ver = SYS_PD_FIRMWARE_VERSION;
固件版本号，默认从 1 开始
gp_usbpd_cfg->hw_ver = SYS_PD_HARDWARE_VERSION;
硬件版本号，默认从 1 开始
gp_usbpd_cfg->port_cfg.spec_rev = SPEC_REVISION_3_0;
Pc 协议版本号
gp_usbpd_cfg->port_cfg.cbl_type = CABLE_TYPE_C
Pd 接口类型

5.1.9 电源能力和 usb 供应商定义配置

5.1.9.1 source 能力配置

功能	配置 src 的供电能力
函数定义	void sys_pd_cfg_src_power_profile_set(u32_t index)
参数常量	Index 供电能力索引，取值为 POWER_PROFILE_0P5W_FIXED，POWER_PROFILE_15W_FIXED, POWER_PROFILE_18W_FIXED, POWER_PROFILE_18W_PPS, POWER_PROFILE_30W_FIXED, POWER_PROFILE_30W_PPS, POWER_PROFILE_45W_FIXED, POWER_PROFILE_45W_PPS, POWER_PROFILE_60W_FIXED, POWER_PROFILE_60W_PPS, POWER_PROFILE_80W_FIXED, POWER_PROFILE_80W_PPS, POWER_PROFILE_100W_FIXED, POWER_PROFILE_100W_PPS,
返回	无

5.1.9.2 usb 设备 id 配置

gp_usbpd_cfg->vd_usb.vid = DEFAULT_USB_VID;

Usb vid 设置

gp_usbpd_cfg->vd_usb.tid = DEFAULT_USB_TID;

Usb tid 设置

gp_usbpd_cfg->vd_usb.bcd_device = DEFAULT_USB_BCD_DEVICE;

Usb bcd 设备 id

gp_usbpd_cfg->vd_usb.prod_id = DEFAULT_USB_PRODUCT_ID;

生产 id

5.1.9.3 固定电源模式 source cap 配置

功能	Sourc cap 配置
函数定义	Void sys_pd_cfg_port_pwr_fixed_cap_set(volatile POWER_FIXED_t *fixed, u32_t volt, u32_t curr, u32_t pwm_duty)
参数常量	Fixed,固定电源；volt，电压；curr，电流；pwm_duty，pwm 占空比
返回	无

5.1.9.4 供电 pwn 配置

gp_usbpd_cfg->port_cfg.power.src.cap_num

Source cap 的数量

gp_usbpd_cfg->port_cfg.power.src.pwm_en

Pwn 供电使能

gp_usbpd_cfg->port_cfg.power.src.pwm_per

Pwn 周期设置

gp_usbpd_cfg->port_cfg.src_pdp

Src 供电功率设置

5.1.9.5 pps mode 供电 src cap 配置

功能	pps mode 供电 src cap 配置
函数定义	void sys_pd_cfg_port_pwr_pps_cap_set(volatile POWER_PPS_t *pps, u32_t min_volt, u32_t max_volt, u32_t max_curr, u32_t pwm_duty, u32_t pwm_duty_max)
参数常量	pps 可变电源；min_volt 最小电压；max_volt 最大电压；max_curr 最大电流；pwm_duty 占空比；pwm_duty_max pwm 最大占空比

返回	无

5.1.9.6 端口 pps 使能

功能	端口 pps 使能，设置 gp_usbpd_cfg->port_cfg.pps_en
函数定义	usbpd_cfg_port_pps_en_set(val)
参数常量	val， 1 打开或者 0 关闭
返回	无

5.1.9.7 获取 pd-src 供电索引值

功能	通过电压来索引 pd-src 供电档位
函数定义	u32_t sys_pd_cfg_src_fixed_power_profile_index_get(u32_t voltage)
参数常量	Voltage=电压
返回	0~6

5.1.9.8 电池模式 src-cap 配置

功能	电源结构体成员赋值；电池模式配置 src-cap 的电压电流范围，电源类型，占空比配置等
函数定义	void sys_pd_cfg_port_pwr_battery_cap_set(volatile POWER_BATTERY_t *battery, u32_t min_volt, u32_t max_volt, u32_t power, u32_t pwm_duty, u32_t pwm_duty_max)
参数常量	Battery，电池类型结构体；min_volt,最小电压；max_volt,最大电压；power，电池最大电压；pwm_duty，占空比；pwm_duty_max，最大占空比
返回	无

5.1.9.9 可编程模式 src-cap 配置

功能	电源结构体成员赋值；可编程电源模式配置电压，电流范围，配置占空比等
函数定义	void sys_pd_cfg_port_pwr_vari_cap_set(volatile POWER_VARIABLE_t *vari, u32_t min_volt, u32_t max_volt, u32_t curr, u32_t pwm_duty, u32_t pwm_duty_max)
参数常量	Vari，可编程电源结构体；min_volt,最小电压；max_volt,最大电压；curr，电流；pwm_duty，占空比；pwm_duty_max，最大占空比
返回	无

5.1.10sink 的受电能力

5.1.10.1 sink 受电设置接口

功能	设置 sink 的受电能力
函数定义	void sys_pd_cfg_snk_operating_power_profile_set(u32_t index)
参数常量	Index 受电能力索引，有效值为以下： SNK_POWER_REQUIRE_15W,SNK_POWER_REQUIRE_SRC_MAX,SNK_POWER_REQUIRE_

	18W, SNK_POWER_REQUIRE_40W, SNK_POWER_REQUIRE_OP5W, SNK_POWER_REQUIRE_24W, SNK_POWER_REQUIRE_29W, SNK_POWER_REQUIRE_30W, SNK_POWER_REQUIRE_100W,
返回	无

5.1.10.2 sink 受电能力设置

gp_usbpd_cfg->port_cfg.power.snk.profile
索引设置
gp_usbpd_cfg->port_cfg.power.snk.opr_volt
单位工作电压设置
gp_usbpd_cfg->port_cfg.power.snk.opr_curr
单位工作电流设置
gp_usbpd_cfg->port_cfg.power.snk.max_opr_curr
单位最大工作电流设置
gp_usbpd_cfg->port_cfg.operational_pdp
单位工作功率设置
gp_usbpd_cfg->port_cfg.maximum_pdp
单位最大工作功率设置

5.2 设备策略管理器（DPM）

设备策略管理器负责接口的连接管理

5.2.1 Type-C 配置通道（CC）

5.2.1.1 更新 cc 通道读出的 adc 值

功能	更新 cc1，cc2 通道的值，并将其分别赋值给 g_dpm_info.cc2_adc_rslt g_dpm_info.cc2_adc_rslt
函数定义	void dpm_cc_update_adc()
参数常量	无
返回	无

5.2.1.2sink 获取 cc1 和 cc2 连接状态

功能	获取 cc1 和 cc2 连接状态，如果连接并设置连接位置
函数定义	bool_t dpm_cc_source_detected()
参数常量	无
返回	0 未连接或者 1 连接

5.2.1.3 src 获取 cc1 和 cc2 的连接状态

功能	src 获取 cc1 和 cc2 的连接状态，如果连接并设置连接位置
函数定义	bool_t dpm_cc_connection_detected()

参数常量	无
返回	无

5.2.1.4 设置 sink 连接成功的状态

功能	设置 sink 设备连接状态，cc 端口状态等
函数定义	void dpm_cc_state_to_attached_snk()
参数常量	无
返回	无

5.2.1.5 vcon 使能上电

功能	vcon 使能上电
函数定义	void dpm_cc_state_cable_plug_detect()
参数常量	无
返回	无

5.2.1.6 配置 src 设备连接后的状态

功能	配置 src 的连接状态，cc 的状态，cc 端口电压
函数定义	void dpm_cc_state_to_attached_src()
参数常量	无
返回	无

5.2.1.7cc 状态初始化

功能	cc 状态初始化
函数定义	void dpm_cc_state_init()
参数常量	无
返回	无

5.2.1.8 cc 连接状态机

功能	cc 连接状态机，时时根据 cc 的状态做相应的处理
函数定义	void dpm_cc_state_machine()
参数常量	无
返回	无

5.2.1.9 获取 cc 的连接状态

功能	获取 cc 的连接状态
函数定义	dpm_cc_attached()
参数常量	无
返回	1 连接或者 0 断开

5.2.1.10 获取 cc 的断开状态

功能	获取 cc 的断开状态
函数定义	dpm_cc_unattached()
参数常量	无
返回	1 断开，0 连接

5.2.1.11 获取 cc 的状态

功能	获取 cc 的状态
函数定义	dpm_cc_state_get()
参数常量	无
返回	0~19

5.2.1.12 设置 cc 的状态

功能	设置 cc 的状态
函数定义	dpm_cc_state_set(v)
参数常量	V=0~19
返回	无

5.2.1.13 设置 vconn 的状态

功能	打开或者关闭 vconn
函数定义	dpm_presently_vconn_source_set(en)
参数常量	en 0 关闭或者 1 打开
返回	无

5.2.1.14 获取 vconn 的状态

功能	获取 vconn 的状态
函数定义	dpm_presently_vconn_source_get()
参数常量	无
返回	无

5.2.1.15 cc1 和 cc2 当前 adc 值设置

功能	cc1 cc2 当前的 adc 设置

函数定义	dpm_cc1_adc_rslt_curr_set(v) dpm_cc2_adc_rslt_curr_set(v)
参数常量	v =adc 值
返回	无

5.2.1.16 获取 cc1 和 cc2 的 adc 的值

功能	获取当下 cc1 和 cc2 的 adc 值
函数定义	dpm_cc1_adc_rslt_curr_get() dpm_cc2_adc_rslt_curr_get()
参数常量	无
返回	当下 cc1 和 cc2 的 adc 值

5.2.1.17 设置 adc 读取次数

功能	设置 adc 读取次数
函数定义	dpm_cc_adc_reading_cnt_set(v)
参数常量	v 读取的次，一般用来清 0
返回	无

5.2.1.18 adc 读取次数加 1

功能	Adc 读取次数加 1
函数定义	dpm_cc_adc_reading_cnt_inc()
参数常量	无
返回	Adc 当下的读取次数

5.2.1.19 获取 adc 的读取次数

功能	获取 adc 的读取次数
函数定义	dpm_cc_adc_reading_cnt_get()
参数常量	无
返回	cc 的 Adc 的读取次数

5.2.1.20 设置 cc1 和 cc2 前两次读取的 adc 值

功能	设置 cc1 和 cc2 前两次读取的 adc 值
函数定义	dpm_cc1_adc_rslt_last1_set(v) dpm_cc1_adc_rslt_last2_set(v) dpm_cc2_adc_rslt_last1_set(v) dpm_cc2_adc_rslt_last2_set(v)
参数常量	v 读取的 adc 值
返回	无

5.2.1.21 获取 cc1 和 cc2 前两次读取的 adc 值

功能	获取 cc1 和 cc2 前两次读取的 adc 值
函数定义	dpm_cc1_adc_rslt_last1_get() dpm_cc1_adc_rslt_last2_get() dpm_cc2_adc_rslt_last1_get() dpm_cc2_adc_rslt_last2_get()
参数常量	无
返回	cc1 和 cc2 前两次的 Adc 的值

5.2.1.22 获取 cc 连接状态改变的时间

功能	获取 cc 连接状态改变的时间
函数定义	dpm_cc_connection_changed_time_get()
参数常量	无
返回	无符号 32 位整数

5.2.1.23 设置 cc 连接状态改变的时间

功能	设置 cc 连接状态改变的时间
函数定义	dpm_cc_connection_changed_time_set(v)
参数常量	V 时间
返回	无

5.2.1.24 设置 cc1 和 cc2 上 vconn 的使能状态

功能	设置 cc1 和 cc2 上 vconn 的使能状态
函数定义	dpm_vconn_en_cc1_set(flg) dpm_vconn_en_cc2_set(flg)
参数常量	Flg 状态标志位 1 使能或者 0 关闭
返回	无

5.2.1.25 获取 cc1 和 cc2 上 vconn 的使能状态

功能	获取 vconn 的状态
函数定义	dpm_vconn_en_cc1_get() dpm_vconn_en_cc2_get()
参数常量	无
返回	1 打开或者 0 关闭

5.2.2 端口策略管理

5.2.2.1 端口状态初始化

功能	g_dpm_info 结构体初始化，cc 端口初始化，捕获模式，插座电缆模式初始化等
函数定义	void dpm_port_init()
参数常量	无
返回	无

5.2.2.2 usb 端口数据电源角色设置

功能	根据当前的端口的电源角色设置数据角色，并更新电源角色
函数定义	status_t dpm_port_role_update()
参数常量	无
返回	设置成功返回 0，设置失败返回-1

5.2.2.3 根据连接状态设置电源的状态

功能	Pe prl 初始化，以及电源的设置
函数定义	void dpm_cable_detect_done_set(u32_t flg)
参数常量	flg 标志位，1 连接状态，0 断开状态
返回	无

5.2.2.4 sop 以及复位类型消息检测设置

功能	设置检测测 sop sop’ sop”，以及复位消息
函数定义	status_t dpm_sop_cbl_type_set(u32_t type)
参数常量	无
返回	0

5.2.2.5 获取 sop 类型消息侦测的设置

功能	获取那种 sop 消息以及复位等消息被侦测
函数定义	u32_t dpm_sop_cbl_type_get()
参数常量	无
返回	32 位无符号整数，其中低 7 位是有效位

5.2.2.6 根据需求输出电压

功能	输出对应的需求电压

函数定义	void dpm_power_supply_req()
参数常量	无
返回	无

5.2.2.7 usb 数据角色交换能力评估

功能	确认当下 usb 设备是否可以进行数据角色交换
函数定义	void dpm_drs_evaluate_req()
参数常量	无
返回	无

5.2.2.8 usb 数据角色变更申请

功能	实现数据角色能力交换，并修改对应的 gpio 口配置
函数定义	void dpm_drs_change_req()
参数常量	无
返回	无

5.2.2.9 pd 电源角色交换评估

功能	评估 pd 变更电源角色的能力
函数定义	void dpm_prs_evaluate_req()
参数常量	无
返回	无

5.2.2.10 pd 电源角色交换申请

功能	更改电源 vbus 状态，进入电源角色交换流程
函数定义	void dpm_prs_change_req(u32_t on)
参数常量	On 等于 1 电源切换到 5v，vbus 使能，等于 0，关闭 vbus
返回	无

5.2.2.11 清除 cc 上 rp 和 rd 的设置

功能	清除 rp 和 rd 的设置
函数定义	void dpm_rp_rd_clear()
参数常量	无
返回	无

5.2.2.12 使能 rd

功能	使能 cc1 和 cc2 上的 rd，用于检测 src 是否连接
函数定义	void dpm_rd_asserted_req()

参数常量	无
返回	无

5.2.2.13 使能 rp

功能	Rp 电流源使能，可以检测 sink 是否连接
函数定义	void dpm_rp_asserted_req()
参数常量	无
返回	无

5.2.2.14 vconn 电源交换评估

功能	评估是否可以进行 vconn 交换
函数定义	void dpm_vcs_evaluate_req()
参数常量	无
返回	无

5.2.2.15 vconn 电源交换申请

功能	实现 vconn 电源交换
函数定义	void dpm_vconn_change_req(u32_t on)
参数常量	on 1 使能 vconn，0 关闭 vconn
返回	无

5.2.2.16 pd 复位

功能	初始化端口状态，cc 连接状态，pe 状态，prl 状态
函数定义	void dpm_port_reset()
参数常量	无
返回	无

5.2.2.17 休眠处理函数

功能	休眠之前对一些状态进行保存，关闭中断
函数定义	bool_t dpm_on_sleep_handler(void)
参数常量	无
返回	1

5.2.2.18 唤醒处理函数

功能	唤醒后恢复休眠之前的状态
函数定义	bool_t dpm_on_wakeup_handler(void)

参数常量	无
返回	1

5.2.2.19 DPM 双载波模式 2 请求

功能	DPM 双载波模式 2 请求
函数定义	void dpm_bist_carrier_mode2_req(u32_t sop_type)
参数常量	Sop_type 消息类型
返回	无

5.2.2.20 DPM bist 测试数据请求

功能	DPM bist 测试数据请求
函数定义	void dpm_bist_test_data_req(u8_t *data_buf, u32_t len, u32_t sop_type)
参数常量	data_buf 数据指针；len 数据长度；Sop_type 消息类型
返回	无

5.2.2.21 获取 DPM bist 测试数据

功能	获取 DPM bist 测试数据
函数定义	u8_t dpm_bist_test_data_get(volatile u8_t *data_buf)
参数常量	data_buf 目标数据指针
返回	无

5.2.2.22 检测 vbus 电压是否降低

功能	检测 vbus 是否降低超过最低要求
函数定义	bool_t dpm_src_ocp_vbus_drop_check()
参数常量	无
返回	1vbus 电压低于要求或者 0vbus 电压满足要求

5.2.2.23 新电压申请

功能	申请新的电压水平
函数定义	bool_t dpm_new_power_level_req()
参数常量	无
返回	1 可以申请新电压，0 电压不能更换

5.2.2.24 获取 vbus 的电压

功能	获取当下 vbus 的电压
函数定义	u32_t dpm_vbus_volt_get()

参数常量	无
返回	32 位无符号整数

5.2.2.25 检测 vbus 是否存在

功能	检测 vbus 连接状态
函数定义	bool_t dpm_vbus_detected()
参数常量	无
返回	1vbus 存在，0vbus 不存在

5.2.2.26 检测 vbus 的电压值

功能	检测输入的电压是不是 vbus 当下的电压
函数定义	bool_t dpm_vbus_volt_check(u32_t volt)
参数常量	Volt =输入电压值
返回	1 是，0 不是

5.2.2.27 检测 vbus 电压是否丢失

功能	用于 sink 监测 vbus 电压，判断 src 是否断开连接
函数定义	bool_t dpm_vbus_drop_check()
参数常量	无
返回	1vbus 丢失，0vbus 连接正常

5.2.2.28 检测 vbus 的 vsave0v 状态

功能	检测 vbus 电压是否为 0v
函数定义	bool_t dpm_vbus_is_at_vsafe0v()
参数常量	无
返回	无

5.2.2.29 端口复位

功能	g_dpm_info 结构体清 0
函数定义	void dpm_info_reset()
参数常量	无
返回	无

5.2.2.30 端口状态标志位设置与获取

dpm_cable_detect_done_get() //获取 cc 连接状态
dpm_not_type_c_set(flg)//设置非 typec 端口模式，输入 1
dpm_not_type_c_get()//获取是不是 type-c 端口
dpm_current_pwr_mode_set(flg)//设置当下的供电角色，0snk，1src

dpm_current_pwr_mode_get()//获取当下供电角色
dpm_dfp_mode_set(flg)//设置 usb 数据角色，1dfp，0ufp
dpm_dfp_mode_get()//获取 usb 数据角色是否为 dfp
dpm_ufp_mode_set(flg)//设置 usb 数据角色为 ufp
dpm_ufp_mode_get()//获取 usb 数据额角色是不是为 ufp
dpm_vcon_is_on_set(flg)//设置 vconn 电源打开状态标志位
dpm_vcon_is_on_get()//获取 vconn 电源的打开状态标志位
dpm_vcon_is_off_set(flg)//设置 vconn 电源关闭标志位
dpm_vcon_is_off_get()//获取 vconn 电源关闭标志位
dpm_source_cap_change_set(flg)//设置 src-cap 更改标志位
dpm_source_cap_change_get()//获取 src-cap 更改标志位
dpm_req_identity_discov_set(flg)//设置请求发送 discover identity 消息标志位
dpm_req_identity_discov_get()//获取请求发送 discover identity 消息标志位
dpm_req_svid_discov_set(flg) //设置请求发送 discover svid 消息标志位
dpm_req_svid_discov_get()//获取请求发送 discover svid 消息标志位
dpm_req_mode_discov_set(flg)//设置请求发送发现模式消息标志位
dpm_req_mode_discov_get()//获取请求发送发现模式消息标志位
dpm_req_mode_enter_set(flg)//设置请求发送进入模式消息标志位
dpm_req_mode_enter_get()//获取请求发送进入模式消息标志位
dpm_req_mode_exit_set(flg)//设置请求发送退出模式消息标志位
dpm_req_mode_exit_get()//获取请求发送退出模式消息标志位
dpm_default_pwr_mode_set(flg)//设置默认供电角色标志位
dpm_default_pwr_mode_get()//获取默认供电角色标志位
dpm_request_can_be_met_set(flg)//设置满足请求消息标志位
dpm_request_can_be_met_get()//获取满足请求消息标志位
dpm_request_cant_be_met_set(flg)//设置不满足请求消息标志位
dpm_request_cant_be_met_get()//获取不满足请求消息标志位
dpm_request_could_be_met_later_set(flg)//设置请求推迟满足标志位
dpm_request_could_be_met_later_get()//获取请求消息推迟满足标志位
dpm_snk_transit_to_new_power_req()//设置 snk 请求新的电压标志位
dpm_power_supply_ready_get()//设置 pd 设备准备好供电标志位
dpm_get_snk_cap_req_set(flg)//设置发送请求 snk 能力消息的标志位
dpm_get_snk_cap_req_get()//获取发送请求 snk 能力消息标志位
dpm_goto_min_req_set(flg)//设置发送 goto min 消息标志位
dpm_goto_min_req_get()//获取发送 goto min 消息标志位
dpm_vbus_at_vsafe5v_set(flg)//设置 vbus 在 vsafe5v 标志位
dpm_vbus_at_vsafe5v_get() //获取 vbus 在 vsafe5v 标志位
dpm_dr_swp_req_set(flg)//设置数据角色交换请求标志位
dpm_dr_swp_req_get()//获取数据角色交换请求标志位

dpm_pr_swp_req_set(flg)//设置电源角色交换请求标志位
dpm_pr_swp_req_get()//获取电源角色交换请求标志位
dpm_prs_req_rejected_set(flg)//设置拒绝电源角色交换申请标志位
dpm_prs_req_rejected_get()//获取拒绝电源角色交换申请标志位
dpm_drs_req_rejected_set(flg)//设置拒绝数据角色交换申请标志位
dpm_drs_req_rejected_get()//获取拒绝数据角色交换申请标志位
dpm_vcs_req_rejected_set(flg)//设置拒绝 vconn 电源交换请求标志位
dpm_vcs_req_rejected_get()//获取拒绝 vconn 电源交换请求标志位
dpm_pr_swp_requested_flag_set(flg)//设置支持电源角色请求交换标志位
dpm_pr_swp_requested_flag_get()//获取支持电源角色申请交换标志位
dpm_dr_swp_requested_flag_set(flg)//设置支持数据角色申请交换标志位

dpm_dr_swp_requested_flag_get()//获取支持数据角色申请交换标志位
dpm_vcon_swp_requested_flag_set(flg)//设置支持 vconn 电源申请交换标志位
dpm_vcon_swp_requested_flag_get()//获取支持 vconn 电源申请交换标志位
dpm_vdm_disc_ident_requested_flag_set(flg)//设置支持请求发送 discover ident 消息标志位
dpm_vdm_disc_ident_requested_flag_get()//获取支持请求发送 discover ident 消息标志位
dpm_get_src_cap_req_set(flg) //设置 src 发送 src-cap 请求消息标志位
dpm_get_src_cap_req_get() //获取 src 发送 src-cap 请求消息标志位
dpm_vconn_swp_req_set(flg) //设置 vconn 交换请求标志位
dpm_vconn_swp_req_get()//获取 vconn 交换请求标志位
dpm_req_attention_frm_ufp_set(flg)//设置请求 ufp 发送 attention 消息标志位
dpm_req_attention_frm_ufp_get()//获取请求 ufp 发送 attention 消息标志位
dpm_contract_valid_set(flg)//设置协议有效标志位
dpm_contract_valid_get()//获取协议有效标志位
dpm_hard_reset_req_set(flg) //设置硬件复位标志位
dpm_hard_reset_req_get()//获取硬件复位标志位
dpm_pwr_snk_at_default_set(flg)//设置是否为 sink 标志位
dpm_pwr_snk_at_default_get()//获取是否为 snk'标志位
dpm_response_received_set(flg)//设置 snk 正确接收 src-cap 标志位
dpm_response_received_get()//获取 snk 是否正确接收 src-cap
dpm_new_pwrlevel_req_set(flg)//设置新电压协商请求标志位
dpm_new_pwrlevel_req_get()//获取新电压协商请求标志位
dpm_req_src_cap_update_set(flg)//设置 snk 重新获取 src-cap 标志位
dpm_req_src_cap_update_get()//获取 snk 重新获取 src-cap 标志位
dpm_dr_swp_ok_set(flg)//设置支持 usb 数据角色交换支持标志位
dpm_dr_swp_ok_get()//获取支持 usb 数据角色交换支持标志位
dpm_dr_swp_not_ok_set(flg) //设置 usb 数据角色交换不支持标志位
dpm_dr_swp_not_ok_get()//获取 usb 数据角色交换不支持标志位
dpm_furthur_evaluate_drswp_is_needed_set(flg)//设置 usb 数据角色交换评估标志位
dpm_furthur_evaluate_drswp_is_needed_get() //获取 usb 数据角色交换评估标志位
dpm_pr_swp_ok_set(flg)//设置支持 pd 设备支持电源角色交换标志位
dpm_pr_swp_ok_get()//获取支持 pd 设备支持电源角色交换标志位
dpm_pr_swp_not_ok_set(flg)//设置电源角色交换失败标志位

dpm_pr_swp_not_ok_get()//获取电源角色交换标志位
dpm_furthur_evaluate_prswp_is_needed_set(flg)//设置支持电源角色交换评估标志位
dpm_furthur_evaluate_prswp_is_needed_get()//获取支持电源角色交换评估标志位
dpm_src_snk_set(flg)//设置设备为 drp 标志位
dpm_src_snk_get()//获取设备为 drp 标志位
dpm_rd_asserted_get()//获取 rd 配置状态
dpm_rp_asserted_get()//获取 rp 的配置状态
dpm_vcon_swp_not_ok_set(flg)//设置 vconn 源不支持交换标志位
dpm_vcon_swp_not_ok_get() //获取支持 vconn 源不支持交换标志位
dpm_vcon_swp_ok_set(flg)//设置支持 vconn 源交换标志位
dpm_vcon_swp_ok_get()//获取 vconn 源支持交换标志位
dpm_furthur_evaluate_vcswp_is_needed_set(flg)//设置 vconn 源交换评估标志位
dpm_furthur_evaluate_vcswp_is_needed_get()//获取 vconn 源交换评估标志位
dpm_mode_entered_set(flg)//设置进入模式标志位
dpm_mode_entered_get()//获取进入模式标志位
dpm_mode_enter_nak_set(flg)//设置进入模式失败标志位

dpm_mode_enter_nak_get() //获取进入模式失败标志位
dpm_mode_exited_set(flg)//设置退出模式标志位
dpm_mode_exited_get()//获取退出模式标志位
dpm_cable_plug_set(flg)//设置电缆插头模式标志位
dpm_cable_plug_get()//获取是否为电缆插头模式
dpm_cable_reset_req_set(flg)//设置电缆申请复位标志位
dpm_cable_reset_req_get()//获取电缆复位申请标志位
dpm_vdm_nak_set(flg) //设置 ufp 对供应商定义消息无响应标志位
dpm_vdm_nak_get()//获取 ufp 对供应商定义消息无响应标志位
dpm_vdm_busy_set(flg)//设置对 vdm 消息传递忙碌标志位
dpm_vdm_busy_get()//获取 vdm 消息传递忙碌标志位
dpm_vdm_ack_set(flg)//设置正确接收 vdm 消息标志位
dpm_vdm_ack_get()//获取正确接收 vdm 消息标志位
dpm_pe_rxmsg_ind_set(flg)//设置 rx 正在接收信息标志位
dpm_pe_rxmsg_ind_get()//获取 rx 正在接收信息标志位
dpm_pe_rxmsg_type_set(flg)//设置接收消息的类型
dpm_pe_rxmsg_type_get()//获取接收消息类型
dpm_src_obj_pos_set(flg)//设置申请的 src-cap 位置
dpm_src_obj_pos_get()//获取申请的 src-cap 的位置
dpm_msg_sop_type_set(flg)//设置收到的消息的类型
dpm_msg_sop_type_get()//获取收到消息的类型
dpm_cbl_plug_detected_get()//获取是否侦测到线缆插头
dpm_cbl_plug_detected_set(flg)//设置侦测到线缆插头标志位
dpm_orient_select_get()//获取 cc 连接位置
dpm_orient_select_set(flg)//设置 cc 连接位置
dpm_curr_mode_set(flg)//设置当下工作模式
dpm_curr_mode_get()//获取当下的工作模式
dpm_vbus_source_en_get()// 获取 vbus 使能状态
dpm_vbus_source_en_set(flg)//使能 vbus

dpm_port_disabled_set(flg)//端口状态设置
dpm_port_disabled_get()//获取端口使能状态
dpm_port_dead_battery_set(flg)//无电电池标志位设置
dpm_port_dead_battery_get()//获取无电电池标志位
dpm_cc1_adc_rslt_set(v)//设置 cc1 读取的 adc 的值
dpm_cc1_adc_rslt_get()//获取 cc1 的 adc 值
dpm_cc2_adc_rslt_set(v)//设置 cc2 读取的 adc 值
dpm_cc2_adc_rslt_get()//获取 cc2 读取的 adc 值
dpm_vdm_sop2_present_set(en)//设置 vdm sop”控制器标志位
dpm_vdm_sop2_present_get()//获取 vdm sop”控制器状态
dpm_tx_in_prog_set(flg)//设置 tx 发送消息过程标志位
pm_tx_in_prog_get()//获取 tx 是否在发送消息
dpm_req_sop1_svid_discov_set(flg)//设置请求发送 discover svid 消息标志位
dpm_req_sop1_svid_discov_get() //获取请求发送 discover svid 消息标志位状态
dpm_req_sop1_mode_discov_set(flg)//设置请求发送 discover mode 消息标志位
dpm_req_sop1_mode_discov_get()//获取请求发送 discover mode 消息标志位状态
dpm_req_sop1_mode_enter_set(flg)//设置请求发送 进入 mode 消息标志位
dpm_req_sop1_mode_enter_get() //获取请求发送进入 mode 消息标志位状态
dpm_req_sop1_mode_exit_set(flg)//设置请求发送 退出 mode 消息标志位
dpm_req_sop1_mode_exit_get()//获取请求发送退出 mode 消息标志位状态
dpm_req_sop2_svid_discov_set(flg)//设置请求发送 discover svid 消息标志位，sop”类型

dpm_req_sop2_svid_discov_get()//获取请求发送 discover svid 消息标志位状态，sop”类型
dpm_req_sop2_mode_discov_set(flg)//设置请求发送 discover mode 消息标志位，sop”类型
dpm_req_sop2_mode_discov_get()//获取请求发送 discover mode 消息标志位状态，sop”类型
dpm_req_sop2_mode_enter_set(flg)//设置请求发送 进入 mode 消息标志位，sop”类型
dpm_req_sop2_mode_enter_get()//获取请求发送进入 mode 消息标志位状态，sop”类型
dpm_req_sop2_mode_exit_set(flg) //设置请求发送 退出 mode 消息标志位，sop”类型
dpm_req_sop2_mode_exit_get()//获取请求发送退出 mode 消息标志位状态，sop”类型
dpm_def_pwr_mode_configured_set(flg)//设置 pd 为电源模式标志位
dpm_def_pwr_mode_configured_get()//获取 pd 是不是在电源模式

5.2.3 QC 接口

5.2.3.1 qc 结构体初始化

功能	qc 结构体初始化
函数定义	void dpm_qc_state_init(volatile DPM_QC_INFO_t *p_qc_info, bool_t snk)
参数常量	p_qc_info =qc 结构体指针；snk=1 是 snk，0 不是
返回	无

5.2.3.2 获取 qc 使能状态

功能	获取 qc 的使能状态
函数定义	bool_t dpm_qc_enable_get(volatile DPM_QC_INFO_t *p_qc_info)
参数常量	p_qc_info =qc 结构体指针
返回	1，qc 使能；0，qc 关闭

5.2.3.3 设置 qc 的使能状态

功能	设置 qc 的使能状态，设置 dmdp mode
函数定义	void dpm_qc_enable_set(volatile DPM_QC_INFO_t *p_qc_info, bool_t on, bool_t snk)
参数常量	p_qc_info =qc 结构体指针；on，使能标志位，1 使能，0 关闭；snk，1 位 snk，0 为 src
返回	无

5.2.3.4 qc 连接状态机

功能	Qc 连接状态机，以及不同状态做不同的处理
函数定义	void dpm_qc_state_machine(volatile DPM_QC_INFO_t *p_qc_info)
参数常量	p_qc_info =qc 结构体指针
返回	无

5.2.3.5 设置或者获取 qc 的状态

功能	设置或者获取 qc 的状态
函数定义	dpm_qc_state_get(ptr) dpm_qc_state_set(ptr,v)
参数常量	Ptr=qc 结构体指针；v=设置状态 0~16
返回	0~16

5.2.3.6 qc dpdm 模式设置

功能	设置 qc dpdm mode 为基本模式或者三星苹果模式，通过宏 DPM_BC_APPLE_SAMSUNG_MODE_ENABLED 来控制
函数定义	dpm_qc_src_io_init(ptr)
参数常量	Prt=qc 结构体指针
返回	无

5.2.3.7 qc -src 和 snk 设置 dpdm mode

功能	设置 src 和 snk 状态 dpdm 模式
函数定义	dpm_qc_src_io_init(ptr) dpm_qc_snk_io_init(ptr)
参数常量	Prt=qc 结构体指针
返回	无

5.3 策略引擎（PE）

5.3.1 pe 状态初始化

功能	g_pe_info 结构体初始化为 0，如果 cc 连接且为 src，启动 t_fst_src_cap_timer 定时器
函数定义	void pe_state_init()
参数常量	无
返回	无

5.3.2 pe 控制发送 tx 消息

功能	设置 dpm tx 消息类型并发送消息
函数定义	void pe_txmsg_req(u32_t msg_type)
参数常量	msg_type,消息类型
返回	无

5.3.3 pe 控制发送 tx 扩展消息

功能	设置 dpm tx 消息类型并发送消息
函数定义	void pe_txmsg_req_ex(u32_t msg_type, u32_t sop_type)
参数常量	msg_type,消息类型
返回	无

5.3.4 pe-prl 复位

功能	协议层复位
函数定义	void pe_prl_reset()
参数常量	无
返回	无

5.3.5 pe 状态机

功能	根据 pe 不同的状态，做出不同的处理
函数定义	void pe_state_machine()
参数常量	无
返回	无

5.3.6 线缆插头模式 pe 状态机

功能	根据 pe 不同的状态，做出不同的处理
函数定义	void pe_cbl_state_machine()
参数常量	无
返回	无

5.3.7 tx 消息发送状态机

功能	根据发送的消息类型，启动对应的定时器
函数定义	void pe_tx_ind_start_timer()
参数常量	无
返回	无

5.3.8 本地硬件复位

功能	本地硬件复位
函数定义	pe_local_hw_reset()
参数常量	无
返回	无

5.4 协议（PRL）

5.4.1 协议层硬件复位策略管理

5.4.1.1 协议层硬件复位处理状态机

功能	处理接收到的硬件复位
函数定义	void prl_hr_state_machine()
参数常量	无
返回	无

5.4.1.2 协议层状态初始化

功能	g_prl_state_info 结构体初始化为 0，并设置 pd 协议版本，发送数据可以尝试的次数
函数定义	void prl_state_init()
参数常量	无
返回	无

5.4.1.3 协议层硬件复位

功能	协议层初始化，tx rx 状态初始化，id 清 0
函数定义	void prl_hr_reset()
参数常量	无
返回	无

5.4.2 协议层 rx 接收数据策略管理

5.4.2.1 协议层 rx 状态初始化

功能	清除消息 id，清除 rx 消息头，设置 rx 的 state 为 PRL_RX_WAIT_FOR_PHY_MSG
函数定义	void prl_rx_state_init()
参数常量	无
返回	无

5.4.2.2 协议层 rx 接收数据状态机

功能	根据不同的 rx 状态，做对应的处理，主要包括接收数据，并解析数据，存储消息 id
函数定义	void prl_rx_state_machine()
参数常量	无

返回	无

5.4.3 协议层 tx 发送数据策略管理

5.4.3.1 协议层 tx 状态初始化

功能	协议层 tx 状态初始化，清除 msg id，清除相关标志位等
函数定义	void prl_tx_state_init()
参数常量	无
返回	无

5.4.3.2 协议层 tx 发送消息

功能	确定消息 id，消息长度，消息类型，然后发送消息
函数定义	void prl_tx_sop_msg_send()
参数常量	无
返回	无

5.4.3.3 协议层 tx 状态机

功能	处理 tx 发送端的各种状态
函数定义	void prl_tx_state_machine()
参数常量	无
返回	无

5.4.4 协议层消息解析器

5.4.4.1 消息收发地址初始化

功能	初始化收发消息的内存地址，对于以下数据头指针进行赋值
函数定义	void prl_msg_parser_init()
参数常量	无
返回	无

5.4.4.2 接收数据拷贝

功能	接受数据从内存空间拷贝到软件指定空间
函数定义	void prl_msg_rx_dma_copy()
参数常量	无
返回	无

5.4.4.3 捕获模式 rx 消息处理

功能	捕获模式 rx 接收消息解析
函数定义	void prl_msg_capture_rx()
参数常量	无
返回	无

5.4.4.4 协议层 rx 消息解析

功能	协议层 rx 消息解析，针对不同的信息进行处理
函数定义	void prl_msg_rx()
参数常量	无
返回	无

5.4.4.5 dma 收发数据起始结束地址设置

功能	Dma 发送数据起始结束地址设置，接收数据起始结束地址设置，以及收发数据起始结束地址获取
函数定义	prl_msg_tx_dma_start_addr_set(v); prl_msg_tx_dma_start_addr_get() prl_msg_tx_dma_end_addr_set(v); prl_msg_tx_dma_end_addr_get() prl_msg_rx_dma_start_addr_set(v); prl_msg_rx_dma_start_addr_get() prl_msg_rx_dma_end_addr_set(v); prl_msg_rx_dma_end_addr_get()
参数常量	V=地址
返回	地址

5.4.4.6 协议层 tx 端数据包组合

功能	组合 tx 数据包
函数定义	void prl_msg_tx(u32_t tx_msg_type, u32_t tx_sop_type)
参数常量	Msg_type =消息类型; sop_type=sop 类型（0~8）
返回	无

5.4.4.7 获取发送消息缓存区地址

功能	获取发送消息缓存区地址，软件维护
函数定义	prl_msg_tx_wptr_get()
参数常量	无
返回	无

5.4.4.8 dma 发送数据缓冲区硬件读取指针地址

功能	dma 发送数据缓冲区硬件读取指针地址
函数定义	prl_msg_tx_rptr_get()

参数常量	无
返回	无

5.4.4.9 dma 接收数据缓冲区硬件写指针地址

功能	dma 接收数据缓冲区硬件写指针地址
函数定义	prl_msg_rx_wptr_get()
参数常量	无
返回	写指针地址

5.4.4.10 pd 接收消息 dma 缓存区读取指针

功能	pd 接收消息 dma 缓存区读取指针
函数定义	prl_msg_rx_rptr_get()
参数常量	无
返回	接收消息写指针地址

5.4.4.11 设置 pd 接收数据读取指针

功能	设置 pd 接收数据读取指针
函数定义	prl_msg_rx_rptr_set(v)
参数常量	V=地址
返回	无

5.4.4.12 获取发送消息描述信息

功能	获取发送消息描述信息，包括的数据包的类型，数据长度等
函数定义	prl_msg_tx_desc_get()
参数常量	无
返回	32 位数据

5.4.4.13 设置发送消息信息描述

功能	设置发送消息信息描述
函数定义	prl_msg_tx_desc_set(v)
参数常量	V=设置的数据
返回	无

5.4.4.14 获取接收数据信息描述

功能	获取接收数据信息描述
函数定义	prl_msg_rx_desc_get()
参数常量	无
返回	32 位描述消息

5.4.4.15 设置接收数据描述信息

功能	设置接收数据描述信息
函数定义	prl_msg_rx_desc_set(v)
参数常量	V=设置数据
返回	无

5.4.4.16 获取接收数据消息头

功能	获取接收数据消息头
函数定义	prl_msg_rx_head_get()
参数常量	无
返回	16 位消息头

5.4.4.17 获取发送数据消息头

功能	获取发送数据消息头
函数定义	prl_msg_tx_head_get()
参数常量	无
返回	16 位消息头

5.4.4.18 获取接收扩展数据消息头

功能	获取扩展数据消息头
函数定义	prl_msg_rx_ext_head_get()
参数常量	无
返回	16 位扩展数据消息头

5.4.4.19 获取接收的扩展有效数据地址

功能	获取接收的扩展有效数据地址
函数定义	prl_msg_rx_ext_data_get()
参数常量	无
返回	数据地址

5.5 系统回调函数

5.5.1 pd 回调函数

5.5.1.1snk 断开回调函数

功能	Snk 断开连接时调用函数，qc 模式，关闭 qc；pd 模式，关闭 pwm 功能

函数定义	void sys_pd_sink_unattached_hook()
参数常量	无
返回	无

5.5.1.2 sink 等待连接回调函数

功能	Sink 等待连接进入低功耗模式
函数定义	void sys_pd_sink_attachwait_hook()
参数常量	无
返回	无

5.5.1.3 sink 连接成功回调函数

功能	Sink 连接成功回调函数
函数定义	void sys_pd_sink_attached_hook()
参数常量	无
返回	无

5.5.1.4 pd-snk 设置新的受电能力回调函数

功能	pd-snk 设置新的受电能力回调函数
函数定义	status_t sys_pd_sink_transit_to_new_power_hook()
参数常量	无
返回	无

5.5.1.5 src 断开连接回调函数

功能	Src 断开连接时调用函数，qc 模式，关闭 qc；pd 模式，关闭 pwm 功能
函数定义	void sys_pd_source_unattached_hook()
参数常量	无
返回	无

5.5.1.6 pd-src 等待连接回调函数

功能	Src 等待连接，进入低功耗模式
函数定义	void sys_pd_source_attachwait_hook()
参数常量	无
返回	无

5.5.1.7 src 连接成功回调函数

功能	Src 连接成功回调函数，qc 模式初始化，pd-src 通过 pwm 开电
函数定义	void sys_pd_source_attached_hook()

参数常量	无
返回	无

5.5.1.8 src 输出新的电压电流

功能	Src 输出新的电压，电流
函数定义	status_t sys_pd_source_transit_to_new_power_hook(u32_t pos)
参数常量	Pos=src cap 索引
返回	成功返回 0，失败返回-1 或者-2

5.5.1.9 vbus 使能状态获取回调函数

功能	Vbus 使能状态获取回调函数
函数定义	bool_t sys_pd_vbus_source_en_get(void)
参数常量	无
返回	0 关闭或者 1 打开

5.5.1.10 vbus 使能设置回调函数

功能	设置 vbus 的使能状态，如果为 pwn 控制宏控打开，输出 5v 电
函数定义	void sys_pd_vbus_source_en_set(bool_t flg)
参数常量	Flg=使能标志，打开 1 或者 0 关闭
返回	无

5.5.1.11 vbus 放电使能状态获取回调函数

功能	Vbus 放电使能状态获取
函数定义	bool_t sys_pd_vbus_discharge_en_get(void)
参数常量	无
返回	1 打开或者 0 关闭

5.5.1.12 设置 vbus 的放电使能状态回调函数

功能	设置 vbus 使能状态
函数定义	void sys_pd_vbus_discharge_en_set(bool_t flg)
参数常量	Flg=标志位，1 打开，0 关闭
返回	无

5.5.1.13 vconn 使能状态获取和设置回调函数

功能	Cc1 cc2 vconn 电源使能设置，以及状态获取
函数定义	bool_t sys_pd_cc1_vconn_en_get(void) void sys_pd_cc1_vconn_en_set(bool_t flg) bool_t sys_pd_cc2_vconn_en_get(void)

	void sys_pd_cc2_vconn_en_set(bool_t flg)
参数常量	Flg=标志位，1 打开或者 0 关闭
返回	0 关闭或者 1 打开

5.5.1.14 电源角色开始交换回调函数

功能	电源角色回调函数
函数定义	void sys_pd_prs_started_hook(u32_t arg)
参数常量	arg=0 或者 1
返回	无

5.5.1.15 电源角色完成交换回调函数

功能	电源角色完成交换回调函数
函数定义	void sys_pd_prs_completed_hook(u32_t arg)
参数常量	Arg=0 未完成或者 1 完成
返回	无

5.5.1.16 数据角色开始交换回调函数

功能	数据角色开始交换回调函数
函数定义	void sys_pd_drs_started_hook(u32_t arg)
参数常量	Arg=0 或者 1
返回	无

5.5.1.17 数据角色完成交换回调函数

功能	数据角色完成交换回调函数
函数定义	void sys_pd_drs_completed_hook(u32_t arg)
参数常量	Arg=0 或者 1
返回	无

5.5.1.18 vconn 开始交换回调函数

功能	Vconn 电流源角色交换开始
函数定义	void sys_pd_vcs_started_hook(u32_t arg)
参数常量	Arg=1 或者 0
返回	无

5.5.1.19 vconn 完成交换回调函数

功能	Vconn 电源角色完成交换处理函数

函数定义	void sys_pd_vcs_completed_hook(u32_t arg)
参数常量	Arg=1 或者 0
返回	无

5.5.1.20 pd 电源配置更改请求回调函数

功能	Pd 电源配置更改请求回调函数
函数定义	void sys_pd_power_change_req_hook(u32_t arg)
参数常量	Arg =0 或者 1
返回	无

5.5.1.21 pd-led 状态设置

功能	pd-led 状态设置
函数定义	void sys_pd_led_state_set(SYS_PD_LED_STATE_t state)
参数常量	State=0~4
返回	无

5.5.1.22 pd idle 状态回调函数

功能	处理 qc 状态机和 100ms 处理系统进程
函数定义	void sys_pd_idle_hook(u32_t arg)
参数常量	Arg=32 位无符号整数
返回	无

5.5.1.23 收到供应商自定义 Discover Identity 消息回调函数

功能	收到 Discover Identity 消息处理
函数定义	void sys_pd_vdm_disc_ident_ack_received_hook(u32_t arg)
参数常量	Arg=0 或者 1
返回	无

5.5.1.24 收到或者发送警告消息回调函数

功能	收到后者发送警告消息处理
函数定义	void sys_pd_prl_msg_alert_hook(UPD_DATA_MSG_ALERT_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=警告消息指针；dir=消息来源，0rx 或者 1tx
返回	无

5.5.1.25 收到或者发送获取国家信息回调函数

功能	收到或者发送获国家信息处理函数
函数定义	void sys_pd_prl_msg_get_country_info_hook(UPD_DATA_MSG_GET_COUNTRY_INFO_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=获取国家信息的控制消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.26 收到或者发送电池状态信息回调函数

功能	收到或者发送电池状态信息处理函数
函数定义	Void sys_pd_prl_msg_battery_status_hook(UPD_DATA_MSG_BATTERY_STATUS_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=电池状态消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.27 收到或者发送扩展 src-cap 回调函数

功能	收到或者发送扩展 src-cap 处理函数
函数定义	void sys_pd_prl_msg_ext_src_cap_hook(UPD_EXT_MSG_SRC_CAP_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展 src-cap 消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.28 收到或者发送扩展状态消息回调函数

功能	收到或者发送扩展状态消息处理函数
函数定义	void sys_pd_prl_msg_ext_status_hook(UPD_EXT_MSG_STATUS_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展状态消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.29 收到或者发送获取扩展电池能力消息回调函数

功能	收到或者发送获取扩展电池能力消息处理函数接口
函数定义	void sys_pd_prl_msg_ext_get_bat_cap_hook(UPD_EXT_MSG_GET_BAT_CAP_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=获取扩展电池能力消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.30 收到或者发送获取扩展电池状态消息回调函数

功能	收到或者发送获取扩展电池状态消息处理函数

函数定义	void sys_pd_prl_msg_ext_get_bat_status_hook(UPD_EXT_MSG_GET_BAT_STATUS_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=获取扩展电池状态消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.31 收到或者发送扩展电池能力消息回调函数

功能	收到或者发送扩展电池能力消息处理
函数定义	void sys_pd_prl_msg_ext_bat_cap_hook(UPD_EXT_MSG_BAT_CAP_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展电池能力消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.32 收到或者发送获取扩展制造商信息回调函数

功能	收到或者发送获取扩展制造商信息处理
函数定义	void sys_pd_prl_msg_ext_get_mfr_info_hook(UPD_EXT_MSG_GET_MFR_INFO_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=获取扩展制造商信息消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.33 收到或者发送扩展制造商信息消息回调函数

功能	收到或者发送扩展制造商信息消息处理
函数定义	void sys_pd_prl_msg_ext_mfr_info_hook(UPD_EXT_MSG_MFR_INFO_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展制造商信息消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.34 收到或者发送扩展响应安全信息回调函数

功能	收到或者发送扩展响应安全信息处理
函数定义	void sys_pd_prl_msg_ext_secur_resp_hook(UPD_EXT_MSG_SECURITY_RESP_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展响应安全信息请求消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.35 收到或者发送扩展请求安全信息消息回调函数

功能	收到或者发送扩展请求安全信息消息处理

函数定义	void sys_pd_prl_msg_ext_secur_req_hook(UPD_EXT_MSG_SECURITY_REQ_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=请求安全信息消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.36 收到或者发送固件升级请求消息回调函数

功能	收到或者发送固件升级请求消息回调函数
函数定义	void sys_pd_prl_msg_ext_fw_up_req_hook(void *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=固件升级请求消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.37 收到或者发送固件升级响应消息回调函数

功能	收到或者发送固件升级响应消息处理
函数定义	void sys_pd_prl_msg_ext_fw_up_resp_hook(void *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=固件升级响应消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.38 收到或者发送扩展 pps 状态信息回调函数

功能	收到或者发送扩展 pps 状态信息处理
函数定义	void sys_pd_prl_msg_ext_pps_status_hook(UPD_EXT_MSG_PPS_STATUS_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展 pps 状态消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.39 收到或者发送扩展国家地区信息消息回调函数

功能	收到或者发送扩展国家地区信息消息处理
函数定义	void sys_pd_prl_msg_ext_country_info_hook(UPD_EXT_MSG_COUNTRY_INFO_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=国家地区消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.40 收到或者发送国家代码回调函数

功能	收到或者发送国家代码处理
函数定义	void

	sys_pd_prl_msg_ext_country_codes_hook(UPD_EXT_MSG_COUNTRY_CODES_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=国家地区代码消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.41 收到或者发送扩展 snk 能力消息回调函数

功能	收到或者发送扩展 snk 能力消息处理
函数定义	void sys_pd_prl_msg_ext_snk_cap_hook(UPD_EXT_MSG_SNK_CAP_t *p_info, PRL_MSG_DIR_TYPE_t dir)
参数常量	P_info=扩展 snk 能力消息指针；dir=消息来源，tx 为 0，rx 为 1
返回	无

5.5.1.42 回调函数接口初始化

功能	回调函数接口初始化
函数定义	void sys_pd_hook_init()
参数常量	无
返回	无

5.5.2 Qc 回调函数

5.5.2.1 qc 接口初始化

功能	Qc 接口初始化
函数定义	void sys_qc_init()
参数常量	无
返回	无

5.5.2.2 qc 使能函数

功能	Qc 功能关闭和打开
函数定义	void sys_qc_disable()
参数常量	无
返回	无

5.5.2.3 获取 usb 电压

功能	获取 usb 的电压
函数定义	u32_t sys_usbc_volt_get(DPM_QC_VOLT_INTF_TYPE_t type)
参数常量	Type 为 0，d+的电压；1，d-电压；2 ，vbus 电压；3 ，不返回电压
返回	电压值，如不是电压输入类型，返回 0

5.5.2.4 设置 vbus 电压

功能	设置 vbus 的输出电压
函数定义	bool_t sys_usbc_vbus_set(u32_t volt)
参数常量	Volt 设置的电压
返回	未成功返回 0，成功返回 1

5.5.2.5 usb 设置 d+和 d-的模式

功能	usb 设置 d+和 d-的模式
函数定义	void sys_usbc_src_dpdm_mode_set(DPM_SRC_DPDM_MODE_t mode)
参数常量	Mode =0~5,0 没有设置任何 mode；1 苹果模式；2 三星模式；3 基本模式 4 qc2.0 模式；5qc3.0 模式
返回	无