超核Kinetis 开发包(固件库)

包含Freescale Kinetis K 系列及 KL 系列底层驱动及实例代码

如何开始

请按步骤逐步学习本固件库及Kinetis,先看免费入门视频教程.该教材包含了你想要的全部内容，从整体知识介绍到工具软件下载到入门提高。

1. 中文入门视频
2. 答疑论坛

所支持的IDE

- Keil(MDK) V5.00 以上版本
- IAR 7.2 以上版本

适用型号

MK60DNxxx MK64F12xxx MK10DN32xxx MKL25Zxxx MKL46Zxxx MK10DZ10 MK20DZ10 MK70F12 MK70F15 MK22F12

- xxx代表任意
- 超核固件库并不仅仅限于以上型号，只要经过简单的移植。出色的兼容性架构设计可以让超核固件库可以运行在几乎所有Kinetis系列芯片上。如需移植服务，请到答疑论坛联系我们

文件说明

- [Document] 文档
- [Libraries] 固件库源代码 启动代码 第三方组件 等等
- [Project] 工程文件及例程 主要以超核K60K-64开发板-渡鸦 为主
  - 超核K60-K64开发板-基础例程: 渡鸦开发板基础例程
  - 超核K60-K64开发板-基础例程：渡鸦开发板拓展例程
  - K60核心板-智能车专用： K60核心板 智能车比赛专用
  - Internal： 超核内部使用，用户不需要可删除
- [**readme.md**] 使用前必读

联系我们及获取最新固件库

- 论坛 www.beyondcore.net
- 技术讨论群:247160311

已经发现的存在问题

1. 一些版本的IAR 不能打开中文路径以及带有空格的路径下的功能，如果遇到这个问题，请将路径改为全英文且没有空格的路径

BUG回报

1. 如果有BUG或者使用问题欢迎邮件咨询 yandld@126.com

4/10/2015 11:18:43 AM

文件列表

这里列出了所有文档化的文件，并附带简要说明:

adc.c	Www.beyondcore.net http://upcmcu.taobao.com
can.c	Www.beyondcore.net http://upcmcu.taobao.com
cmp.c	Www.beyondcore.net http://upcmcu.taobao.com
common.c	Www.beyondcore.net http://upcmcu.taobao.com
cpuidy.c	Www.beyondcore.net http://upcmcu.taobao.com
crc.c	Www.beyondcore.net http://upcmcu.taobao.com
dac.c	Www.beyondcore.net http://upcmcu.taobao.com
dma.c	Www.beyondcore.net http://upcmcu.taobao.com
enet.c	Www.beyondcore.net http://upcmcu.taobao.com
flash.c	Www.beyondcore.net http://upcmcu.taobao.com
flexbus.c	Www.beyondcore.net http://upcmcu.taobao.com
ftm.c	Www.beyondcore.net http://upcmcu.taobao.com
gpio.c	Www.beyondcore.net http://upcmcu.taobao.com
i2c.c	Www.beyondcore.net http://upcmcu.taobao.com
i2s.c	Www.beyondcore.net http://upcmcu.taobao.com
lptmr.c	Www.beyondcore.net http://upcmcu.taobao.com
nfc.c	Www.beyondcore.net http://upcmcu.taobao.com
pdb.c	Www.beyondcore.net http://upcmcu.taobao.com
pit.c	Www.beyondcore.net http://upcmcu.taobao.com
rtc.c	Www.beyondcore.net http://upcmcu.taobao.com
sd.c	Www.beyondcore.net http://upcmcu.taobao.com
spi.c	Www.beyondcore.net http://upcmcu.taobao.com
systick.c	Www.beyondcore.net http://upcmcu.taobao.com
tsi.c	Www.beyondcore.net http://upcmcu.taobao.com
uart.c	Www.beyondcore.net http://upcmcu.taobao.com
vref.c	Www.beyondcore.net http://upcmcu.taobao.com
wdog.c	Www.beyondcore.net http://upcmcu.taobao.com

adc.c 文件参考

浏览源代码.

函数

uint8_t	**ADC_IsConversionCompleted** (uint32_t instance, uint32_t mux)
int32_t	**ADC_Calibration** (uint32_t instance)
void	**ADC_Init** (ADC_InitTypeDef *ADC_InitStruct)
void	**ADC_EnableHardwareTrigger** (uint32_t instance, bool status)
uint8_t	**ADC_QuickInit** (uint32_t MAP, ADC_ResolutionMode_Type resolutionMode)
void	**ADC_StartConversion** (uint32_t instance, uint32_t chl, uint32_t mux)
void	**ADC_ChlMuxConfig** (uint32_t instance, uint32_t mux)
int32_t	**ADC_ReadValue** (uint32_t instance, uint32_t mux)
int32_t	**ADC_QuickReadValue** (uint32_t MAP)
void	**ADC_ITDMAConfig** (uint32_t instance, uint32_t mux, ADC_ITDMAConfig_Type config)
void	**ADC_CallbackInstall** (uint32_t instance, ADC_CallBackType AppCBFun)
void	**ADC0_IRQHandler** (void)
void	**ADC1_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.25
2015.9.26 FreeXc 完善了adc.c & adc.h文件的注释

注解
此文件为芯片ADC模块的底层功能函数

在文件 **adc.c** 中定义.

函数说明

void ADC0_IRQHandler (void)

中断处理函数入口

ADC0_IRQHandler :芯片的ADC0模块中断函数入口

注解
函数内部用于中断事件处理，用户无需使用

在文件 adc.c 第 445 行定义.

void ADC1_IRQHandler (void)

中断处理函数入口

ADC1_IRQHandler :芯片的ADC0模块中断函数入口

注解

函数内部用于中断事件处理,用户无需使用

在文件 adc.c 第 458 行定义.

int32_t ADC_Calibration (uint32_t instance)

AD采集校准函数，内部函数,用户无需调用

参数

[in] **instance** ADC 模块号

- HW_ADC0 :ADC0模块
- HW_ADC1 :ADC1模块
- HW_ADC2 :ADC2模块

返回值

0 校准完成

1 校准失败

在文件 adc.c 第 94 行定义.

void ADC_CallbackInstall (uint32_t instance, ADC_CallBackType AppCBFun)

注册中断回调函数

参数

[in] **instance** ADC 模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **AppCBFun** 回调函数指针入口

返回值

None

参见

对于此函数的具体应用请查阅应用实例

在文件 adc.c 第 432 行定义.

void ADC_ChlMuxConfig (uint32_t instance, uint32_t mux)

ADC Mux select.

```
1 //启动 ADC0 通道在A模式下进行数据转换
2 ADC_ChlMuxConfig(HW_ADC0, kADC_MuxA);
```

参数

[in] **instance** ADC模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **mux** ADC转换器通道复用选择

- kADC_MuxA A转换器触发
- kADC_MuxB B转换器触发

返回值

None

在文件 **adc.c** 第 **323** 行定义.

void ADC_EnableHardwareTrigger (uint32_t instance, bool status)

AD采集硬件/软件触发选择

参数

[in] **instance** ADC 模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **status** Conversion trigger select

- 0 Software trigger selected
- 1 Hardware trigger selected

返回值

None

在文件 **adc.c** 第 **233** 行定义.

void ADC_Init (ADC_InitTypeDef * ADC_InitStruct)

ADC模块工作初始化配置

```
1  //使用adc0模块的1通道 单端模式 8位精度 软件触发
2  ADC_InitTypeDef ADC_InitStruct1;  //申请一个结构体
3  ADC_InitStruct1.chl = 1;  //1通道
4  ADC_InitStruct1.clockDiv = kADC_ClockDiv8; //ADC转换时钟为输入时钟(默认BusClock) 的8分频，和转换速度相关
5  ADC_InitStruct1.instance = HW_ADC0;         //选择ADC0模块
6  ADC_InitStruct1.resolutionMode = kADC_SingleDiff8or9; //单端模式下8位精度 查分模式下9位精度
7  ADC_InitStruct1.SingleOrDifferential = kADC_Single;    //选择单端模式
8  ADC_InitStruct1.triggerMode = kADC_TriggleSoftware;    //设置为软件触发
9  ADC_InitStruct1.vref = kADC_VoltageVREF;               //使用外部VERFH VREFL 作为模拟电压参考
10 //初始化ADC模块
11 ADC_Init(&ADC_InitStruct1);
```

参数

[in] **ADC_InitStruct** ADC初始化结构体，内容详见注释

返回值

None

在文件 adc.c 第 155 行定义.

uint8_t ADC_IsConversionCompleted (uint32_t instance, uint32_t mux)

判断AD转换是否结束

```
1  //查看ADC0 模块的A通道的转换是否完成
2  ADC_IsConversionCompleted(HW_ADC0, kADC_MuxA);
```

参数

[in] **instance** ADC 模块号

- HW_ADC0 :ADC0模块
- HW_ADC1 :ADC1模块
- HW_ADC2 :ADC2模块

[in] **mux** ADC 转换器通道 复用 选择

- kADC_MuxA :A通道模式
- kADC_MuxB :B通道模式

返回值

0 转换完成

1 转换未完成

在文件 adc.c 第 75 行定义.

void ADC_ITDMAConfig (uint32_t instance, uint32_t mux, ADC_ITDMAConfig_Type config

)

ADC中断及DMA功能开关函数

```
1 //配置AD0模块 转换完成中断
2 ADC_ITDMAConfig(HW_ADC0, kADC_MuxA, kADC_IT_EOF);
```

参数

[in] **instance** ADC模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **mux** ADC转换器通道复用选择

- kADC_MuxA A通道模式
- kADC_MuxB B通道模式

[in] **config** ADC中断及DMA配置

- kADC_IT_Disable 关闭中断
- kADC_DMA_Disable 关闭DMA功能
- kADC_IT_EOF 转换完成中断
- kADC_DMA_EOF DMA完成中断

返回值

None

在文件 **adc.c** 第 **399** 行定义.

uint8_t ADC_QuickInit (uint32_t MAP,
ADC_ResolutionMode_Type resolutionMode
)

快速完成AD初始化配置

```
1 //初始化 ADC0 通道20 引脚DM1 单端 精度 12位
2 ADC_QuickInit(ADC0_SE20_DM1, kADC_SingleDiff12or13);
3 //读取AD转换结果
4 value = ADC_QuickReadValue(ADC0_SE20_DM1);
```

参数

[in] **MAP** 快速初始化宏，详见ADC.H文件

[in] **resolutionMode** 转换分辨率设置

- kADC_SingleDiff8or9 转换精度为8/9位
- kADC_SingleDiff10or11 转换精度为10/11位
- kADC_SingleDiff12or13 转换精度为12/13位
- kADC_SingleDIff16 转换精度为16位

返回值

ADC模块号

在文件 adc.c 第 256 行定义.

int32_t ADC_QuickReadValue (uint32_t MAP)

读取ADC转换结果(简化版) 只需填入ADC快速初始化宏即可

注解

阻塞式 直到数据转换完成

```
1 //读取AD0模块20通道DM1引脚的转换结果
2 uint32_t value;    //存储数据转换结果
3 value = ADC_QuickReadValue(ADC0_SE20_DM1);
```

参数

MAP 快速初始化宏，详见ADC.H文件

返回值

转换结果

在文件 adc.c 第 368 行定义.

int32_t ADC_ReadValue (uint32_t instance, uint32_t mux)

读取ADC转换数据

注解

立即返回 非阻塞式

```
1 //读取 ADC0模块下的在A模式下数据转换结果
2 uint32_t data;    //存储转换结果
3 data = ADC_ReadValue(HW_ADC0, kADC_MuxA);
```

参数

[in] **instance** ADC 模块号

- HW_ADC0 :ADC0模块
- HW_ADC1 :ADC1模块
- HW_ADC2 :ADC2模块

[in] **mux** ADC 转换器通道 复用 选择

- kADC_MuxA :A通道模式
- kADC_MuxB :B通道模式

返回值

读取结果 如果当前还未完成转换 则返回上一次结果

在文件 adc.c 第 352 行定义.

void ADC_StartConversion (uint32_t instance, uint32_t chl, uint32_t mux

)

ADC开始一次转换

注解

立即返回 非阻塞式 不等待转换结果

```
1 //启动 ADC0 通道20 在A模式下数据转换
2 ADC_StartConversion(HW_ADC0, 20, kADC_MuxA);
```

参数

[in] **instance** ADC 模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **chl** ADC 通道号

[in] **mux** ADC 转换器通道 复用 选择

- kADC_MuxA A转换器触发
- kADC_MuxB B转换器触发

返回值

None

在文件 **adc.c** 第 **302** 行定义.

can.c 文件参考

浏览源代码.

函数

void	**CAN_SetRxFilterMask** (uint32_t instance, uint32_t mb, uint32_t mask)
void	**CAN_SetRxMB** (uint32_t instance, uint32_t mb, uint32_t id)
void	**CAN_Init** (CAN_InitTypeDef *Init)
uint32_t	**CAN_QuickInit** (uint32_t CANxMAP, uint32_t baudrate)
uint32_t	**CAN_WriteData** (uint32_t instance, uint32_t mb, uint32_t id, uint8_t *buf, uint8_t len)
uint32_t	**CAN_WriteRemote** (uint32_t instance, uint32_t mb, uint32_t id, uint8_t len)
void	**CAN_CallbackInstall** (uint32_t instance, CAN_CallBackType AppCBFun)
void	**CAN_ITDMAConfig** (uint32_t instance, uint32_t mb, CAN_ITDMAConfig_Type config)
uint32_t	**CAN_ReadData** (uint32_t instance, uint32_t mb, uint32_t *id, uint8_t *buf, uint8_t *len)
uint32_t	**CAN_ReadFIFO** (uint32_t instance, uint32_t *id, uint8_t *buf, uint8_t *len)
bool	**CAN_IsRxFIFOEnable** (uint32_t instance)
void	**CAN_SetRxFIFO** (uint32_t instance)
void	**CAN_IRQHandler** (uint32_t instance)
void	**CAN0_ORed_Message_buffer_IRQHandler** (void)
void	**CAN1_ORed_Message_buffer_IRQHandler** (void)

变量

CAN_Type *const	**CANBase** [] = CAN_BASES

详细描述

作者
YANDLD

版本
V2.5

日期
2014.4.10
2015.10.08 FreeXc 完善了 can 模块的相关注释

在文件 **can.c** 中定义.

函数说明

void CAN0_ORed_Message_buffer_IRQHandler (void)

CAN0中断处理函数入口

注解
函数内部用于中断事件处理

在文件 can.c 第 555 行定义.

void CAN1_ORed_Message_buffer_IRQHandler (void)

CAN1中断处理函数入口

注解
函数内部用于中断事件处理

在文件 can.c 第 561 行定义.

void CAN_CallbackInstall (uint32_t instance, CAN_CallBackType AppCBFun)

注册中断回调函数

参数
[in] **instance** CAN模块中断入口号

- HW_CAN0 芯片的CAN模块0中断入口
- HW_CAN1 芯片的CAN模块1中断入口

[in] **AppCBFun** 回调函数指针入口

返回值
None

注解
对于此函数的具体应用请查阅应用实例

在文件 can.c 第 374 行定义.

void CAN_Init (CAN_InitTypeDef * Init)

CAN通讯初始化配置 （需要配合使用）

注解
通信速度是基于bus时钟为50MHz时候的计算

参数
[in] **Init** CAN通信模块初始化配置结构体(指针)

参见
CAN_QuickInit() and can.h中相关定义

返回值
none

在文件 can.c 第 178 行定义.

void CAN_IRQHandler (uint32_t instance)

CAN中断处理函数

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

注解

触发中断之后调用用户注册的中断处理函数

在文件 can.c 第 543 行定义.

bool CAN_IsRxFIFOEnable (uint32_t instance)

CAN的FIFO接收是否使能

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

返回值

0 Rx FIFO not enabled

1 Rx FIFO enabled

在文件 can.c 第 504 行定义.

void CAN_ITDMAConfig (uint32_t instance, uint32_t mb, CAN_ITDMAConfig_Type config)

设置CAN模块的中断类型或者DMA功能

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

[in] **mb** 邮箱编号 0~15

[in] **config** 配置模式

- kCAN_IT_Tx_Disable 禁止发送中断
- kCAN_IT_Rx_Disable 禁止接收中断
- kCAN_IT_Tx 发生中断
- kCAN_IT_RX 接收中断

返回值

None

在文件 can.c 第 395 行定义.

---

uint32_t CAN_QuickInit (uint32_t CANxMAP,
uint32_t baudrate
)

CAN通信快速初始化配置

注解

通信速度是基于bus时钟为50MHz时候的计算

参数

[in] **CANxMAP** CAN通信快速配置地图，详见can.h文件
[in] **baudrate** CAN通信波特率

返回值

CAN模块号0或1

在文件 can.c 第 240 行定义.

---

uint32_t CAN_ReadData (uint32_t instance,
uint32_t mb,
uint32_t * id,
uint8_t * buf,
uint8_t * len
)

读取CAN邮箱接收到的数据

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

[in] **mb** CAN通信接收邮箱0~15
[in] **id** CAN通信接收ID指针地址
[in] **buf** CAN通信接收数据指针地址
[in] **len** CAN通信接收数据长度指针地址

返回值

0 正常
1 无数据
2 正在接收

在文件 can.c 第 431 行定义.

---

uint32_t CAN_ReadFIFO (uint32_t instance,

```
                                uint32_t * id,
                                uint8_t *  buf,
                                uint8_t *  len
                               )
```

读取CAN的FIFO中数据

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

[in] **id** CAN通信接收ID指针地址

[in] **buf** CAN通信接收数据指针地址

[in] **len** CAN通信接收数据长度指针地址

返回值

0 正常

在文件 **can.c** 第 **474** 行定义.

void CAN_SetRxFIFO (uint32_t instance)

设置CAN的FIFO接收功能

参数

[in] **instance** CAN模块号

- HW_CAN0 芯片的CAN模块0号
- HW_CAN1 芯片的CAN模块1号

返回值

None

在文件 **can.c** 第 **516** 行定义.

void CAN_SetRxFilterMask (uint32_t instance, uint32_t mb, uint32_t mask)

设置CAN通讯屏蔽掩码

注解

内部函数 用于设置邮箱过滤的

参数

[in] **instance** CAN通信模块号

- HW_CAN0 0号CAN通信模块
- HW_CAN1 1号CAN通信模块

[in] **mb** CAN通信接收邮箱0~15
[in] **mask** CAN通信接收过滤掩码

返回值
none

在文件 **can.c** 第 **137** 行定义.

---

void CAN_SetRxMB (uint32_t instance,
uint32_t mb,
uint32_t id
)

设置CAN通讯接收邮箱

注解
用户调用函数

参数
[in] **instance** CAN通信模块号

- HW_CAN0 0号CAN通信模块
- HW_CAN1 1号CAN通信模块

[in] **mb** CAN通信接收邮箱0~15
[in] **id** CAN通信接收ID，11位标准地址或者28位扩展地址

返回值
none

在文件 **can.c** 第 **164** 行定义.

---

uint32_t CAN_WriteData (uint32_t instance,
uint32_t mb,
uint32_t id,
uint8_t * buf,
uint8_t len
)

CAN send data frame.

参数
[in] **instance** can instance

- HW_CAN0
- HW_CAN1

[in] **mb** Rx MessageBox
[in] **id** the Rx ID(if use filter)
[in] **buf** Rx buffer pointer
[in] **len** Rx frame len

返回值

0 ok
other err

在文件 **can.c** 第 **290** 行定义.

uint32_t CAN_WriteRemote (uint32_t instance,
uint32_t mb,
uint32_t id,
uint8_t len
)

CAN send remote frame.

注解
CAN remote frame has no data section, but has len section

参数
[in] **instance** can instance

- HW_CAN0
- HW_CAN1

[in] **mb** Rx MessageBox
[in] **id** the Rx ID(if use filter)
[in] **len** Rx frame len

返回值
0 ok
other err

在文件 **can.c** 第 **338** 行定义.

cmp.c 文件参考

浏览源代码.

函数

void **CMP_Init** (CMP_InitTypeDef *CMP_InitStruct)

详细描述

作者
YANDLD

版本
V2.5

日期
2015.1.26

在文件 **cmp.c** 中定义.

common.c 文件参考

浏览源代码.

函数

uint32_t	**GetClock** (Clock_t clockName)
void	**EnterSTOPMode** (bool enSleepOnExit)
uint32_t	**QuickInitEncode** (map_t *type)
uint32_t	**GetUID** (void)
void	**QuickInitDecode** (uint32_t map, map_t *type)
void	**DWT_DelayInit** (void)
void	**DWT_DelayUs** (uint32_t us)
void	**DWT_DelayMs** (uint32_t ms)
void	**DelayInit** (void)
void	**DelayMs** (uint32_t ms)
void	**DelayUs** (uint32_t us)
void	**assert_failed** (char *file, uint32_t line)
void	**NMI_isr** (void)
void	**NMI_Handler** (void)
void	**HardFault_Handler** (void)
void	**BusFault_Handler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2013.12.25
2015.10.04 FreeXc完善了common.h & common.c文件的注释

注解
此文件为内部文件，用户无需调用和修改

在文件 **common.c** 中定义.

函数说明

```
void assert_failed ( char *     file,
                     uint32_t  line
                   )
```

断言检测

参数

[in] **file** 指向字符串的指针

[in] **line** 当前错误处对应的行数

在文件 **common.c** 第 **220** 行定义.

void DelayInit (void)

延时初始化函数

```
1 // 完成延时初始化配置，
2 //使用内核的Systick模块实现延时功能
3 DelayInit();
```

返回值

None

在文件 **common.c** 第 **174** 行定义.

void DelayMs (uint32_t ms)

抽象毫秒级的延时设置函数

```
1 // 实现500ms的延时功能
2 DelayMs(500);
```

参数

[in] **ms** 需要延时的时间，单位毫秒

返回值

None

注解

首先需要完成延时初始化配置

在文件 **common.c** 第 **190** 行定义.

void DelayUs (uint32_t us)

抽象微秒级的延时设置函数

```
1 // 实现500us的延时功能
2 DelayUs(500);
```

参数

[in] **us** 需要延时的时间，单位微秒

返回值

None

注解

首先需要完成延时初始化配置

在文件 common.c 第 206 行定义.

void DWT_DelayInit (void)

DWT delay function.

返回值

None

在文件 common.c 第 122 行定义.

void DWT_DelayMs (uint32_t ms)

DWT毫秒级延时

参数

[in] **ms** 延时毫秒数

返回值

None

在文件 common.c 第 159 行定义.

void DWT_DelayUs (uint32_t us)

DWT微秒级延时

参数

[in] **us** 延时微秒数

注解

DWT(Data and Address Watchpoints)具有一个Core Clock的计数器，通过该计数器来实现us级延时

返回值

None

在文件 common.c 第 137 行定义.

void EnterSTOPMode (bool enSleepOnExit)

进入低功耗模式

参数

[in] **enSleepOnExit** 在系统唤醒时候 是否继续进入低功耗

返回值

None

注解

任何中断 都可以唤醒CPU

在文件 common.c 第 69 行定义.

uint32_t GetClock (Clock_t clockName)

获得系统各个总线时钟的频率

```
1 //获得总线时钟频率
2 printf("BusClock:%dHz\r\n", GetClock(kBusClock));
```

参数

[in] **clockName** 时钟名称

- kCoreClock 内核时钟
- kSystemClock 系统时钟 = 内核时钟
- kBusClock 总线时钟
- kFlexBusClock Flexbus总线时钟
- kFlashClock Flash总线时钟

返回值

0 成功
非0 错误

在文件 common.c 第 38 行定义.

uint32_t GetUID (void)

获得芯片UID信息(全球唯一识别码)

返回值

UID信息

在文件 common.c 第 96 行定义.

void QuickInitDecode (uint32_t map, map_t * type)

解码快速初始化结构 用户不需调用

参数

[in] **map** 32位快速初始化编码
[out] **type** 快速初始化结构指针

返回值

None

在文件 common.c 第 112 行定义.

uint32_t QuickInitEncode (map_t * type)

编码快速初始化结构 用户不需调用

参数

[in] **type** 快速初始化结构体指针

返回值

32位快速初始化编码

在文件 **common.c** 第 **87** 行定义.

cpuidy.c 文件参考

浏览源代码.

函数

char *	**CPUIDY_GetFamID** (void)
uint32_t	**CPUIDY_GetPinCount** (void)
uint32_t	**CPUIDY_GetMemSize** (CPUIDY_MemSize_Type memSizeName)
void	**CPUIDY_GetUID** (uint32_t *UIDArray)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.24
2015.10.04 FreeXc 完善了cpuidy.c & cpuidy.h文件的注释

注解
此文件为内部文件，用于获取芯片的出厂信息，少部分用户使用

在文件 **cpuidy.c** 中定义.

函数说明

char* CPUIDY_GetFamID (void)

获得芯片的系列ID

```
1 // 打印芯片型号
2 printf("Family Type:%s\r\n", CPUIDY_GetFamID());
```

返回值
ID字符串指针

在文件 **cpuidy.c** 第 **62** 行定义.

uint32_t CPUIDY_GetMemSize (CPUIDY_MemSize_Type memSizeName)

获得芯片ROM/RAM 大小

```
1 // 获取芯片Pflash的大小并显示出来
2 uint32_t PFlashSize;
3 PFlashSize = CPUIDY_GetMemSize(kPFlashSizeInKB);
4 printf("PFlash Size:%dKB\r\n", PFlashSize);
```

参数

[in] **memSizeName** 存储器类型

- kPFlashSizeInKB 编程Flash的尺寸
- kDFlashSizeInKB 数据Flash的尺寸
- kFlexNVMSizeInKB FlexNVM的尺寸
- kEEPROMSizeInByte EEPOROM的尺寸
- kRAMSizeInKB 芯片RAM的尺寸

返回值

None

在文件 **cpuidy.c** 第 **99** 行定义.

uint32_t CPUIDY_GetPinCount (void)

获得芯片的引脚数

```
1  // 获取芯片的引脚数
2  uint32_t PinCnt;
3  PinCnt = CPUIDY_GetPinCount();
4  //将芯片的引脚数显示出来
5  printf("Pin Cnt:%d\r\n", PinCnt);
```

返回值

引脚数量

在文件 **cpuidy.c** 第 **78** 行定义.

void CPUIDY_GetUID (uint32_t * UIDArray)

获得芯片UID信息(全球唯一识别码)

```
1  // 获取芯片的UID并显示出来
2  uint32_t UID[4],i;
3  CPUIDY_GetUID(UID);
4  for(i = 0; i < 4; i++)
5  {
6      printf("UID[%d]:0x%x\n", i, UID[i]);
7  }
```

参数

[in] **UIDArray** UID数据的首地址

返回值

None

在文件 **cpuidy.c** 第 **144** 行定义.

crc.c 文件参考

浏览源代码.

函数

uint16_t	**CRC16_GenerateSoftware** (const uint8_t *src, uint32_t len)
void	**CRC_Init** (CRC_InitTypeDef *CRC_InitStruct)
void	**CRC_QuickInit** (CRC_ProtocolType type)
uint32_t	**CRC_Generate** (uint8_t *data, uint32_t len)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.08 FreeXc 完善了对 crc 模块的相关注释

注解
此文件为芯片IIC模块的底层功能函数

在文件 **crc.c** 中定义.

函数说明

uint32_t CRC_Generate (uint8_t * data, uint32_t len)

计算并产生CRC运算结果

参数
[in] **data** 数据指针
[in] **len** 数据长度

返回
CRC计算结果

在文件 **crc.c** 第 **161** 行定义.

void CRC_Init (CRC_InitTypeDef * CRC_InitStruct)

初始化CRC硬件模块

参数

[in] **CRC_InitStruct** 指向CRC初始化结构体的指针

返回值

None

在文件 **crc.c** 第 **69** 行定义.

void CRC_QuickInit (CRC_ProtocolType type)

快速初始化CRC硬件模块

参数

[in] **type** CRC协议类型，详细请见crc.h文件中的定义

返回值

None

在文件 **crc.c** 第 **108** 行定义.

dac.c 文件参考

浏览源代码.

函数

void	**DAC_Init** (DAC_InitTypeDef *DAC_InitStruct)
uint32_t	**DAC_GetBufferReadPointer** (uint32_t instance)
void	**DAC_SetBufferReadPointer** (uint32_t instance, uint32_t value)
void	**DAC_SetBufferUpperLimit** (uint32_t instance, uint32_t value)
void	**DAC_ITDMAConfig** (uint32_t instance, DAC_ITDMAConfig_Type config)
void	**DAC_SoftwareStartConversion** (uint32_t instance)
void	**DAC_SetWaterMark** (uint32_t instance, DAC_WaterMarkMode_Type value)
void	**DAC_SetBufferValue** (uint32_t instance, uint16_t *buf, uint8_t len)
void	**DAC_CallbackInstall** (uint8_t instance, DAC_CallBackType AppCBFun)
void	**DAC0_IRQHandler** (void)
void	**DAC1_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.03 FreeXc 完善了dac.c&adc.h中API函数的注释

在文件 **dac.c** 中定义.

函数说明

void DAC0_IRQHandler (void)

中断处理函数入口

DAC0_IRQHandler 芯片的DAC0模块中断函数入口

注解
函数内部用于中断事件处理,用户无需使用

在文件 **dac.c** 第 **331** 行定义.

void DAC1_IRQHandler (void)

中断处理函数入口

DAC1_IRQHandler 芯片的DAC1模块中断函数入口

注解
函数内部用于中断事件处理,用户无需使用

在文件 dac.c 第 342 行定义.

void DAC_CallbackInstall (uint8_t instance, DAC_CallBackType AppCBFun)

注册中断回调函数

参数
[in] **instance** DAC 模块号

- HW_DAC0 DAC0模块
- HW_DAC1 DAC1模块

[in] **AppCBFun** 回调函数指针入口

返回值
None

注解
对于此函数的具体应用请查阅应用实例

在文件 dac.c 第 290 行定义.

uint32_t DAC_GetBufferReadPointer (uint32_t instance)

获得DAC模块buffer的指针

参数
[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

注解
返回DAC当前转换到的 buffer 指针 位置

返回值
当前buffer指针位置 0-15

在文件 dac.c 第 117 行定义.

void DAC_Init (DAC_InitTypeDef * DAC_InitStruct)

初始化DAC模块

```
1 DAC_InitTypeDef DAC_InitStruct = {0};
2 DAC_InitStruct.bufferMode = kDAC_Buffer_Swing;
3 DAC_InitStruct.instance = HW_DAC0;
4 DAC_InitStruct.referenceMode = kDAC_Reference_2;
5 DAC_InitStruct.triggerMode = kDAC_TriggerSoftware;
6 DAC_Init(&DAC_InitStruct);
```

参数

[in] **DAC_InitStruct** DAC 初始化结构体

返回值

None

在文件 **dac.c** 第 **49** 行定义.

---

void DAC_ITDMAConfig (uint32_t instance,
DAC_ITDMAConfig_Type config
)

设置DAC模块中断和DMA

参数

[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

[in] **config** 配置选项

- kDAC_DMA_Disable 禁止DAC DMA功能
- kDAC_IT_Disable 禁止DAC 中断功能
- kDAC_IT_BufferPointer_WaterMark 开启DAC 水位中断
- kDAC_IT_BufferPointer_TopFlag 开启DAC ReadPointer = 0中断
- kDAC_IT_BufferPointer_BottomFlag 开始DAC ReadPointer = UpLimit 中断
- kDAC_DMA_BufferPointer_WaterMark 开启DAC 水位中断（DMA）
- kDAC_DMA_BufferPointer_TopFlag 开启DAC ReadPointer = 0中断（DMA）
- kDAC_DMA_BufferPointer_BottomFlag 开始DAC ReadPointer = UpLimit 中断（DMA）

返回值

None

在文件 **dac.c** 第 **171** 行定义.

---

void DAC_SetBufferReadPointer (uint32_t instance,
uint32_t value
)

设置DAC模块buffer的指针

参数

[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

value 指针位置 0-15

返回值

None

在文件 dac.c 第 130 行定义.

---

void DAC_SetBufferUpperLimit (uint32_t instance,
uint32_t value
)

设置DAC模块读取buffer指针时的最高上限值

```
1 //设置DAC0模块buffer指针上限值为2
2 DAC_SetBufferUpperLimit(HW_DAC0, 2);
```

参数

[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

[in] **value** 指针位置上限 0-15

返回值

None

在文件 dac.c 第 148 行定义.

---

void DAC_SetBufferValue (uint32_t instance,
uint16_t * buf,
uint8_t len
)

填充DAC 缓冲区数据

```
1 //填充待转换的数据至缓冲区
2 uint16_t dacVol[3] = {0x400,0x800,0xb00};
3 DAC_SetBufferValue(HW_DAC0, dacVol,3);
```

注意

填充至缓冲区的数据应与设置的转换序列长度相符

参数

[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

[in] **buf** 指向待转换数据的指针

[in] **len** 待转换数据的长度

返回值
None

在文件 dac.c 第 271 行定义.

void DAC_SetWaterMark (uint32_t instance, DAC_WaterMarkMode_Type value)

设置DAC Buffer的水位

参数
[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

value 水位值 0-15

返回值
None

在文件 dac.c 第 235 行定义.

void DAC_SoftwareStartConversion (uint32_t instance)

软件触发DAC开始工作

```
1 //软件触发DAC0开始工作
2 DAC_SoftwareStartConversion(HW_DAC0);
```

参数
[in] **instance** 模块号

- HW_DAC0 芯片的DAC0 模块
- HW_DAC1 芯片的DAC1 模块

注解
will only trigger once and will advence the read pointer by one step

返回值
None

在文件 dac.c 第 222 行定义.

dma.c 文件参考

浏览源代码.

函数

uint32_t	**_DMA_ChlAlloc** (void)
void	**DMA_ChlFree** (uint32_t chl)
uint32_t	**DMA_Init** (DMA_InitTypeDef *DMA_InitStruct)
uint32_t	**DMA_ChlAlloc** (void)
uint32_t	**DMA_GetMajorLoopCount** (uint8_t chl)
void	**DMA_SetMajorLoopCounter** (uint8_t chl, uint32_t val)
void	**DMA_EnableRequest** (uint8_t chl)
void	**DMA_DisableRequest** (uint8_t chl)
void	**DMA_EnableAutoDisableRequest** (uint8_t chl, bool flag)
void	**DMA_EnableMajorLink** (uint8_t chl, uint8_t linkChl, bool flag)
void	**DMA_ITConfig** (uint8_t chl, DMA_ITConfig_Type config, bool status)
void	**DMA_CallbackInstall** (uint8_t chl, DMA_CallBackType AppCBFun)
uint8_t	**DMA_IsMajorLoopComplete** (uint8_t chl)
void	**DMA_SetDestAddress** (uint8_t ch, uint32_t address)
uint32_t	**DMA_GetDestAddress** (uint8_t ch)
void	**DMA_SetSourceAddress** (uint8_t ch, uint32_t address)
uint32_t	**DMA_GetSourceAddress** (uint8_t ch)
void	**DMA_CancelTransfer** (void)
void	**DMA0_IRQHandler** (void)
void	**DMA1_IRQHandler** (void)
void	**DMA2_IRQHandler** (void)
void	**DMA3_IRQHandler** (void)
void	**DMA4_IRQHandler** (void)
void	**DMA5_IRQHandler** (void)
void	**DMA6_IRQHandler** (void)
void	**DMA7_IRQHandler** (void)
void	**DMA8_IRQHandler** (void)
void	**DMA9_IRQHandler** (void)
void	**DMA10_IRQHandler** (void)
void	**DMA11_IRQHandler** (void)
void	**DMA12_IRQHandler** (void)
void	**DMA13_IRQHandler** (void)
void	**DMA14_IRQHandler** (void)
void	**DMA15_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.08 FreeXc 完善了对 dma 模块的相关注释

注解
此文件为芯片DMA模块的底层功能函数

在文件 **dma.c** 中定义.

函数说明

void DMA_CallbackInstall (uint8_t chl, DMA_CallBackType AppCBFun)

注册中断回调函数

参数
[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **AppCBFun** 回调函数指针

返回值
None

在文件 **dma.c** 第 **334** 行定义.

void DMA_CancelTransfer (void)

取消DMA模块指定通道的数据传输

返回值
None

在文件 **dma.c** 第 **439** 行定义.

uint32_t DMA_ChlAlloc (void)

DMA通道分配

```
1       // init DMA
2       uint8_t dma_chl;
3   *   dma_chl = DMA_ChlAlloc();
```

返回
dma channel

在文件 dma.c 第 129 行定义.

void DMA_ChlFree (uint32_t chl)

DMA通道释放

参数
[in] **chl** DMA通道号

返回
None

在文件 dma.c 第 154 行定义.

void DMA_DisableRequest (uint8_t chl)

禁止通道响应传输

```
1  //禁止DMA 的0通道响应传输
2  DMA_DisableRequest(HW_DMA_CH0);
```

参数
[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

返回值
None

在文件 dma.c 第 222 行定义.

void DMA_EnableAutoDisableRequest (uint8_t chl, bool flag)

在Majloop 结束后 是否自动关闭Request

参数
[in] **chl** DMA通道号

- HW_DMA_CH0

- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **flag** enable or disable

- 0 not affect
- 1 automatically clear

在文件 dma.c 第 238 行定义.

void DMA_EnableMajorLink (uint8_t chl, uint8_t linkChl, bool flag)

使能Major LoopLink 功能

注解

当一个通道结束MajorLoopLink后 自动开始另一个通道的传输

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **linkChl** 需要连接到通道号

[in] **flag** enable or disable

- 0 disable
- 1 enable

返回值

None

在文件 dma.c 第 264 行定义.

void DMA_EnableRequest (uint8_t chl)

使能通道响应传输

```
1 //开启DMA 的0通道进行数据传输
2 DMA_EnableRequest(HW_DMA_CH0);
```

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2

- HW_DMA_CH3

返回值

None

在文件 dma.c 第 204 行定义.

uint32_t DMA_GetDestAddress (uint8_t ch)

获取DMA模块指定通道的目标地址

参数

[in] **ch** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

返回值

32位的目标数据地址

在文件 dma.c 第 401 行定义.

uint32_t DMA_GetMajorLoopCount (uint8_t chl)

获得 DMA MajorLoopCount 计数值

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

返回值

计数值

在文件 dma.c 第 168 行定义.

uint32_t DMA_GetSourceAddress (uint8_t ch)

获取DMA模块指定通道的源地址

参数

[in] **ch** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2

- HW_DMA_CH3

返回值

32位的源数据地址

在文件 dma.c 第 430 行定义.

uint32_t DMA_Init (DMA_InitTypeDef * DMA_InitStruct)

初始化DMA模块

参数

[in] **DMA_InitStruct** 指向DMA初始化配置结构体的指针，详见dma.h

返回

分配到的DMA通道

在文件 dma.c 第 57 行定义.

uint8_t DMA_IsMajorLoopComplete (uint8_t chl)

检测DMA传输是否完成

```
1 //检测DMA的0通道是否完成数据传输
2 status = IsMajorLoopComplete(HW_DMA_CH0);
```

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

返回值

0 数据传输完成

1 数据传输未完成

在文件 dma.c 第 357 行定义.

void DMA_ITConfig (uint8_t chl, DMA_ITConfig_Type config, bool status)

设置DMA传输完成中断

```
1 //开启DMA 的0通道的传输完成中断功能
2 DMA_StartTransfer(HW_DMA_CH0);
```

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **config** 配置模式

- kDMA_IT_Half_Disable 禁止DMA传输一半中断触发
- kDMA_IT_Major_Disable 禁止DMA传输完成中断触发
- kDMA_IT_Half 开启DMA传输一半中断触发
- kDMA_IT_Major 开启DMA传世完成中断触发

[in] **status** enable or disable

- 0 disable
- 1 enable

返回值

None

在文件 dma.c 第 301 行定义.

void DMA_SetDestAddress (uint8_t ch,
uint32_t address
)

设置DMA模块指定通道的目标地址

参数

[in] **ch** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **address** 32位的目标数据地址

返回值

None

在文件 dma.c 第 387 行定义.

void DMA_SetMajorLoopCounter (uint8_t chl,
uint32_t val
)

设置 DMA MajorLoopCount 计数值

参数

[in] **chl** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **val** 计数值

返回值

None

注解

数值不能超过 DMA_CITER_ELINKNO_CITER_MASK

在文件 **dma.c** 第 **184** 行定义.

---

void DMA_SetSourceAddress (uint8_t ch,
uint32_t address
)

设置DMA模块指定通道的源地址

参数

[in] **ch** DMA通道号

- HW_DMA_CH0
- HW_DMA_CH1
- HW_DMA_CH2
- HW_DMA_CH3

[in] **address** 32位的源数据地址

返回值

None

在文件 **dma.c** 第 **416** 行定义.

enet.c 文件参考

浏览源代码.

函数

void	**ENET_MII_Init** (void)
bool	**ENET_MII_Write** (uint16_t phy_addr, uint16_t reg_addr, uint16_t data)
bool	**ENET_MII_Read** (uint16_t phy_addr, uint16_t reg_addr, uint16_t *data)
void	**ENET_Init** (ENET_InitTypeDef *ENET_InitStrut)
void	**ENET_MacSendData** (uint8_t *data, uint16_t len)
uint32_t	**ENET_GetReceiveLen** (void)
uint16_t	**ENET_MacReceiveData** (uint8_t *data)
void	**ENET_ITDMAConfig** (ENET_ITDMAConfig_Type config)
void	**ENET_CallbackTxInstall** (ENET_CallBackTxType AppCBFun)
void	**ENET_CallbackRxInstall** (ENET_CallBackRxType AppCBFun)
bool	**ENET_IsTxTransferComplete** (void)
void	**ENET_Transmit_IRQHandler** (void)
void	**ENET_Receive_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.08 FreeXc 完善了enet模块的相关注释

在文件 **enet.c** 中定义.

函数说明

void ENET_CallbackRxInstall (ENET_CallBackRxType AppCBFun)

设置ENET接收中断回调函数

参数
[in] **AppCBFun** 回调函数指针

返回值
None

在文件 **enet.c** 第 **459** 行定义.

void ENET_CallbackTxInstall (ENET_CallBackTxType AppCBFun)

设置ENET发送中断回调函数

参数

[in] **AppCBFun** 回调函数指针

返回值

None

在文件 **enet.c** 第 **446** 行定义.

void ENET_Init (ENET_InitTypeDef * ENET_InitStrut)

初始化以太网模块

注解

用户调用函数

参数

[in] **ENET_InitStrut** 以太网初始化结构指针，详见应用例程

返回值

None

在文件 **enet.c** 第 **255** 行定义.

bool ENET_IsTxTransferComplete (void)

查看以太帧否发送完成

返回值

0 完成
1 未完成

在文件 **enet.c** 第 **472** 行定义.

void ENET_ITDMAConfig (ENET_ITDMAConfig_Type config)

配置ENET模块的中断或者DMA属性

参数

[in] **config** 模式选择

- kENET_IT_TXF_Disable 禁止发送一帧以太网数据帧中断
- kENET_IT_RXF_Disable 禁止接收一帧以太网数据帧中断
- kENET_IT_TXF 发送一帧以太网数据中断
- kENET_IT_RXF 接收一帧以太网数据中断

返回值

None

在文件 enet.c 第 418 行定义.

uint16_t ENET_MacReceiveData (uint8_t * data)

接收一帧以太帧数据

注解
用户调用函数

参数
[in] **data** 数据指针

返回值
接收到的数据长度

在文件 enet.c 第 389 行定义.

void ENET_MacSendData (uint8_t * data, uint16_t len)

发送一帧以太帧数据

注解
用户调用函数

参数
[in] **data** 发送数据指针
[in] **len** 数据长度 (< 1500字节)

返回值
None

在文件 enet.c 第 352 行定义.

void ENET_MII_Init (void)

初始化以太网 MII配置层接口

返回值
None

在文件 enet.c 第 141 行定义.

bool ENET_MII_Read (uint16_t phy_addr, uint16_t reg_addr, uint16_t * data)

读以太网MII配置层数据

参数

[in] **phy_addr** PHY芯片地址
[in] **reg_addr** 寄存器在PHY内部的偏移地址
[in] **data** 需要读入的数据地址

返回值

true
false

在文件 enet.c 第 209 行定义.

bool ENET_MII_Write (uint16_t phy_addr, uint16_t reg_addr, uint16_t data)

写入以太网MII配置层数据

参数

[in] **phy_addr** PHY芯片地址
[in] **reg_addr** 寄存器在PHY内部的偏移地址
[in] **data** 需要写入的数据

返回值

0 成功
其它 失败

在文件 enet.c 第 163 行定义.

void ENET_Receive_IRQHandler (void)

ENET接收中断处理函数

注解

此函数内部用于调用注册的回调函数，用户无需使用

在文件 enet.c 第 499 行定义.

void ENET_Transmit_IRQHandler (void)

ENET发送中断处理函数

注解

此函数内部用于调用注册的回调函数，用户无需使用

在文件 enet.c 第 486 行定义.

flash.c 文件参考

浏览源代码.

类型定义

typedef void(* **flash_run_entry_t**) (volatile uint8_t *reg)

函数

uint32_t	**FLASH_GetSectorSize** (void)
void	**FLASH_Init** (void)
uint8_t	**FLASH_EraseSector** (uint32_t addr)
uint8_t	**FLASH_WriteSector** (uint32_t addr, const uint8_t *buf, uint32_t len)
uint32_t	**FLASH_SetcorTest** (uint32_t addr)
uint32_t	**FLASH_Test** (uint32_t addr, uint32_t len)

变量

volatile uint8_t	**s_flash_command_run** [] = {0x00, 0xB5, 0x80, 0x21, 0x01, 0x70, 0x01, 0x78, 0x09, 0x06, 0xFC, 0xD5,0x00, 0xBD}
flash_run_entry_t	**s_flash_run_entry**

详细描述

作者
YANDLD

版本
V3.0.0

日期
2015.8.28

在文件 **flash.c** 中定义.

函数说明

uint8_t FLASH_EraseSector (uint32_t addr)

flash erase sector

注解
this function will erase a flash sector

参数
addr start addr

返回值
Flash return code

在文件 flash.c 第 101 行定义.

uint32_t FLASH_GetSectorSize (void)

get flash sector size

返回值

flash sector size

在文件 flash.c 第 84 行定义.

uint32_t FLASH_Test (uint32_t addr, uint32_t len)

flash self test

注解

make sure to have enough stack size

返回值

FLASH_OK succ, other flash test fail

在文件 flash.c 第 237 行定义.

uint8_t FLASH_WriteSector (uint32_t addr, const uint8_t * buf, uint32_t len)

flash write sector

注解

len must = sector size

参数

addr start addr
buf : buffer pointer
len : len

返回值

Flash return code

在文件 flash.c 第 131 行定义.

flexbus.c 文件参考

浏览源代码.

函数

void	**FLEXBUS_Init** (FLEXBUS_InitTypeDef *FLEXBUS_InitStruct)
void	**FLEXBUS_PortMuxConfig** (FLEXBUS_PortMultiplexingSelect_Type group, uint32_t config)
void	**FLEXBUS_AdvancedConfig** (uint32_t CS, FLEXBUS_AdvancedConfigTypeDef *FLEXBUS_AdvancedConfigStruct)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.08 FreeXc 完善了对 flexbus 模块的相关注释

注解
此文件为芯片FlexBus模块的底层功能函数

在文件 **flexbus.c** 中定义.

函数说明

void FLEXBUS_AdvancedConfig (uint32_t CS, FLEXBUS_AdvancedConfigTypeDef * FLEXBUS_AdvancedConfigStruct)

高级Flexbus 配置选项

注解
具体的配置应用详见关于FlexBus的使用例程

参数
[in] **CS** 片选通道信号
[in] **FLEXBUS_AdvancedConfigStruct**

返回值
None

在文件 **flexbus.c** 第 **117** 行定义.

void FLEXBUS_Init (FLEXBUS_InitTypeDef * FLEXBUS_InitStruct)

初始化FlexBus模块

注解
具体的配置应用详见关于FlexBus的使用例程

参数

[in] **FLEXBUS_InitStruct** 指向FlexBus初始化配置结构体的指针，详见FlexBus.h

返回值

None

在文件 **flexbus.c** 第 **23** 行定义.

---

void FLEXBUS_PortMuxConfig (FLEXBUS_PortMultiplexingSelect_Type group,
uint32_t config
)

管脚复用配置

参数

[in] **group** 组号

[in] **config** multiplex control config

参见

Reference manual –> Chip select port multiplexing control register (FB_CSPMCR)

在文件 **flexbus.c** 第 **80** 行定义.

ftm.c 文件参考

浏览源代码.

函数

void	**FTM_PWM_InvertPolarity** (uint32_t instance, uint8_t chl, uint32_t config)
uint32_t	**FTM_QD_QuickInit** (uint32_t MAP, FTM_QD_PolarityMode_Type polarity, FTM_QD_Mode_Type mode)
void	**FTM_QD_GetData** (uint32_t instance, int *value, uint8_t *direction)
void	**FTM_QD_ClearCount** (uint32_t instance)
uint8_t	**FTM_PWM_QuickInit** (uint32_t MAP, FTM_PWM_Mode_Type mode, uint32_t req)
void	**FTM_PWM_ChangeDuty** (uint32_t instance, uint8_t chl, uint32_t pwmDuty)
void	**FTM_IC_QuickInit** (uint32_t MAP, FTM_ClockDiv_Type ps)
void	**FTM_IC_SetTriggerMode** (uint32_t instance, uint32_t chl, FTM_IC_Mode_Type mode)
uint32_t	**FTM_GetChlCounter** (uint32_t instance, uint32_t chl)
void	**FTM_SetMoudleCounter** (uint32_t instance, uint32_t val)
void	**FTM_CallbackInstall** (uint32_t instance, FTM_CallBackType AppCBFun)
void	**FTM_ITDMAConfig** (uint32_t instance, FTM_ITDMAConfig_Type config, bool flag)
bool	**FTM_IsChnInterupt** (uint32_t instance, uint32_t chl)
void	**FTM_IRQHandler** (uint32_t instance)
void	**FTM0_IRQHandler** (void)
void	**FTM1_IRQHandler** (void)
void	**FTM2_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.25
2015.9.26 FreeXc 完善了ftm.c & ftm.h 文件的注释

注解
此文件为芯片FTM模块的底层功能函数

在文件 **ftm.c** 中定义.

函数说明

void FTM_CallbackInstall (uint32_t instance,

FTM_CallBackType AppCBFun
)

设置FTM回调函数

```
1          // 注册FTM1模块的中断回调函数FTM1_ISR,名字可任取
2          static void FTM1_ISR(void)
3  *      {
4                ;//用户程序
5          }
6          FTM_CallbackInstall(HW_FTM1, FTM1_ISR);
```

参数

[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定

参数

[in] **AppCBFun** 回调函数指针

返回值

None

在文件 **ftm.c** 第 **653** 行定义.

uint32_t FTM_GetChlCounter (uint32_t instance,
uint32_t chl
)

获得FTM通道计数值

```
1  // 获得FTM1模块通道1的计数值
2  uint32_t InputCaptureValue;
3  InputCaptureValue = FTM_GetChlCounter(HW_FTM1, HW_FTM_CH1);
```

参数

[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

参数

[in] **chl** 通道

- HW_FTM_CH0 通道0
- HW_FTM_CH1 通道1
- HW_FTM_CH2 通道2
- HW_FTM_CH3 通道3
- HW_FTM_CH4 通道4
- HW_FTM_CH5 通道5
- HW_FTM_CH6 通道6
- HW_FTM_CH7 通道7

注意

chl的可输入的参数视不同芯片而定,例如FTM1模块就没有通道7

返回值

通道Counter值

在文件 **ftm.c** 第 **609** 行定义.

void FTM_IC_QuickInit (uint32_t MAP,
FTM_ClockDiv_Type ps
)

初始化FTM 输入捕捉功能

```
1 // 快速初始化FTM1模块通道1的输入捕捉功能，128分频
2 FTM_IC_QuickInit(FTM1_CH1_PA09, kFTM_ClockDiv128);
```

参数

[in] **MAP** 快速初始化通道列表

[in] **ps** 分频,详细请见FTM_ClockDiv_Type枚举类型

返回值

None

在文件 **ftm.c** 第 **513** 行定义.

void FTM_IC_SetTriggerMode (uint32_t instance,
uint32_t chl,
FTM_IC_Mode_Type mode
)

设置输入捕捉触发模式

```
1 // 设置FTM1模块通道1的输入捕捉模式为下降沿中断
2 FTM_IC_SetTriggerMode(HW_FTM1, HW_FTM_CH1,
  kFTM_IC_FallingEdge);
```

参数

[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意
instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

参数
[in] **chl** 通道

- HW_FTM_CH0 通道0
- HW_FTM_CH1 通道1
- HW_FTM_CH2 通道2
- HW_FTM_CH3 通道3
- HW_FTM_CH4 通道4
- HW_FTM_CH5 通道5
- HW_FTM_CH6 通道6
- HW_FTM_CH7 通道7

注意
chl的可输入的参数视不同芯片而定,例如FTM1模块就没有通道7

参数
[in] **mode** 触发模式

- kFTM_IC_FallingEdge 下降沿触发
- kFTM_IC_RisingEdge 上升沿触发
- kFTM_IC_RisingFallingEdge 跳变沿触发

返回值
None

在文件 **ftm.c** 第 **562** 行定义.

bool FTM_IsChnInterupt (uint32_t instance, uint32_t chl)

判断FTM某个通道是否产生中断（输入捕捉模式）

```
1 // 判断FTM0的0通道是否产生中断
2 bool flag;
3 flag = FTM_IsChnInterupt(HW_FTM0, HW_FTM_CH0)
```

参数
[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意
instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

参数
[in] **chl** 通道

- HW_FTM_CH0 通道0
- HW_FTM_CH1 通道1
- HW_FTM_CH2 通道2
- HW_FTM_CH3 通道3
- HW_FTM_CH4 通道4
- HW_FTM_CH5 通道5
- HW_FTM_CH6 通道6
- HW_FTM_CH7 通道7

注意
chl的可输入的参数视不同芯片而定,例如FTM1模块就没有通道7

返回值
0 对应通道没有产生中断
1 对用通道产生中断

在文件 **ftm.c** 第 **736** 行定义.

```
void FTM_ITDMAConfig ( uint32_t                instance,
                       FTM_ITDMAConfig_Type    config,
                       bool                    flag
                     )
```

FTM中断DMA控制

参数
[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意
instance的可输入的参数视不同芯片而定

参数
[in] **config** 控制参数，详细请见FTM_ITDMAConfig_Type的枚举类型
[in] **flag** 中断/DMA使能或者禁止

- 0 disable
- 1 enable

返回值
None

在文件 **ftm.c** 第 **675** 行定义.

```
void FTM_PWM_ChangeDuty ( uint32_t  instance,
                          uint8_t   chl,
                          uint32_t  pwmDuty
```

)

更改指定引脚的PWM波形占空比

```
1 //设置FTM0模块的3通道的PWM波形占空比为50%
2 FTM_PWM_ChangeDuty(HW_FTM0, HW_FTM_CH3, 5000);
```

参数

[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

参数

[in] **chl** 通道

- HW_FTM_CH0 通道0
- HW_FTM_CH1 通道1
- HW_FTM_CH2 通道2
- HW_FTM_CH3 通道3
- HW_FTM_CH4 通道4
- HW_FTM_CH5 通道5
- HW_FTM_CH6 通道6
- HW_FTM_CH7 通道7

注意

chl的可输入的参数视不同芯片而定,例如FTM1模块就没有通道7

参数

pwmDuty 占空比 = pwmDuty/10000*100%

返回值

None

在文件 **ftm.c** 第 **478** 行定义.

uint8_t FTM_PWM_QuickInit (uint32_t MAP,
FTM_PWM_Mode_Type mode,
uint32_t req
)

快速配置初始化FTM模块实现PWM功能

```
1 //设置FTM0模块的3通道在PTA6引脚中产生1000HZ的pwm波形，默认50%占空比
2 FTM_PWM_QuickInit(FTM0_CH3_PA06, kPWM_EdgeAligned, 1000);
```

参数

[in] **MAP** FTM工作在PWM模式下的编码，详见ftm.h文件

[in] **mode** PWM波形输出模式

- kPWM_EdgeAligned 边沿对齐 最常用
- kPWM_Combine 组合模式
- kPWM_Complementary 互补模式 类似组合模式 但是Chl(n) 和 Chl(n+1) 是互补输出

[in] **req** FTM工作频率设置

返回值

None

在文件 **ftm.c** 第 **385** 行定义.

void FTM_QD_ClearCount (uint32_t instance)

复位FTM模块的计数值（清零）

参数

[in] **instance** FTM模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

返回值

None

在文件 **ftm.c** 第 **184** 行定义.

void FTM_QD_GetData (uint32_t instance, int * value, uint8_t * direction)

获得正交解码的数据

参数

[in] **instance** FTM模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定，例如K60没有FTM3模块

参数

[out] **value** 脉冲数据存储地址

[out] **direction** 脉冲方向存储地址

返回值

None

在文件 **ftm.c** 第 **168** 行定义.

---

uint32_t FTM_QD_QuickInit (uint32_t MAP,
FTM_QD_PolarityMode_Type polarity,
FTM_QD_Mode_Type mode
)

快速配置初始化FTM模块实现正交解码功能

参数

[in] **MAP** FTM工作在正交解码模式下的编码，详见ftm.h文件

[in] **polarity** QD 正交解码设置

- kFTM_QD_NormalPolarity 正常极性
- kFTM_QD_InvertedPolarity 反正极性

[in] **mode** QD模式选择

- kQD_PHABEncoding 使用AB相编码器
- kQD_CountDirectionEncoding 使用方向-脉冲型编码器

返回

FTM模块号

在文件 **ftm.c** 第 **109** 行定义.

---

void FTM_SetMoudleCounter (uint32_t instance,
uint32_t val
)

设置FTM主计数Counter值

```
1 // reset the value of the FTM1 counter
2 FTM_SetMoudleCounter(HW_FTM1, 0);
```

参数

[in] **instance** 模块号

- HW_FTM0 FTM0模块
- HW_FTM1 FTM1模块
- HW_FTM2 FTM2模块
- HW_FTM3 FTM3模块

注意

instance的可输入的参数视不同芯片而定

参数

[in] **val** counter value

返回值

None

在文件 **ftm.c** 第 **629** 行定义.

gpio.c 文件参考

浏览源代码.

函数

void	**PORT_PinMuxConfig** (uint32_t instance, uint8_t pin, PORT_PinMux_Type pinMux)
void	**PORT_PinPullConfig** (uint32_t instance, uint8_t pin, PORT_Pull_Type pull)
void	**PORT_PinOpenDrainConfig** (uint32_t instance, uint8_t pin, bool status)
void	**PORT_PinPassiveFilterConfig** (uint32_t instance, uint8_t pin, bool status)
void	**GPIO_PinConfig** (uint32_t instance, uint8_t pin, GPIO_PinConfig_Type mode)
void	**GPIO_Init** (GPIO_InitTypeDef *GPIO_InitStruct)
uint8_t	**GPIO_QuickInit** (uint32_t instance, uint32_t pinx, GPIO_Mode_Type mode)
void	**GPIO_WriteBit** (uint32_t instance, uint8_t pin, uint8_t data)
void	**GPIO_SetBit** (uint32_t instance, uint32_t pin)
void	**GPIO_ResetBit** (uint32_t instance, uint32_t pin)
uint8_t	**GPIO_ReadBit** (uint32_t instance, uint8_t pin)
void	**GPIO_ToggleBit** (uint32_t instance, uint8_t pin)
uint32_t	**GPIO_ReadPort** (uint32_t instance)
void	**GPIO_WritePort** (uint32_t instance, uint32_t data)
void	**GPIO_ITDMAConfig** (uint32_t instance, uint8_t pin, GPIO_ITDMAConfig_Type config, bool status)
void	**GPIO_CallbackInstall** (uint32_t instance, GPIO_CallBackType AppCBFun)
void	**PORTA_IRQHandler** (void)
void	**PORTB_IRQHandler** (void)
void	**PORTC_IRQHandler** (void)
void	**PORTD_IRQHandler** (void)
void	**PORTE_IRQHandler** (void)
void	**PORTF_IRQHandler** (void)

详细描述

作者

YANDLD

版本

V2.5

日期

2014.3.24

2015.9.25 FreeXc 完善了gpio.h & gpio.c文件的注释

注解

gpio driver source file

在文件 gpio.c 中定义.

函数说明

void GPIO_CallbackInstall (uint32_t instance, GPIO_CallBackType AppCBFun)

注册中断回调函数

```
1 //注册PTB10的EXTI中断回调函数
2 void PTB10_EXTI_ISR(uint32_t pinxArray)
3 {
4     if(pinxArray & (1u << 10))
5         ;//用户自定义函数功能
6 }
7 GPIO_CallbackInstall(instance,PTB10_EXTI_ISR);
```

注意

该回调函数是有形参的

参数

[in] **instance** GPIO模块中断入口号

- HW_GPIOA 芯片的PORTA端口中断入口
- HW_GPIOB 芯片的PORTB端口中断入口
- HW_GPIOC 芯片的PORTC端口中断入口
- HW_GPIOD 芯片的PORTD端口中断入口
- HW_GPIOE 芯片的PORTE端口中断入口

[in] **AppCBFun** 回调函数指针入口

返回值

None

参见

对于此函数的具体应用请查阅应用实例

在文件 gpio.c 第 506 行定义.

void GPIO_Init (GPIO_InitTypeDef * GPIO_InitStruct)

GPIO初始化配置

```
1 //初始化配置PORTB端口的10引脚为推挽输出引脚
2 GPIO_InitTypeDef GPIO_InitStruct1;       //申请一个结构变量
3 GPIO_InitStruct1.instance = HW_GPIOB;    //选择PORTB端口
4 GPIO_InitStruct1.mode = kGPIO_Mode_OPP; //推挽输出
5 GPIO_InitStruct1.pinx = 10;              //选择10引脚
6 //调用初始化GPIO函数
7 GPIO_Init(&GPIO_InitStruct1);
```

参数

[in] **GPIO_InitStruct** GPIO初始化结构体，包含了引脚状态参数

参见

GPIO初始化配置例程

返回值

None

在文件 gpio.c 第 185 行定义.

```
void GPIO_ITDMAConfig ( uint32_t                instance,
                        uint8_t                 pin,
                        GPIO_ITDMAConfig_Type   config,
                        bool                    status
                      )
```

设置GPIO引脚中断类型或者DMA功能

```
1 //设置PORTB端口的10引脚为下降沿触发中断
2 GPIO_ITDMAConfig(HW_GPIOB, 10, kGPIO_IT_FallingEdge, true);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

[in] **config** 配置模式

- kGPIO_DMA_RisingEdge DMA上升沿触发
- kGPIO_DMA_FallingEdge DMA下降沿触发
- kGPIO_DMA_RisingFallingEdge DMA上升和下降沿都触发
- kGPIO_IT_Low 低电平触发中断
- kGPIO_IT_RisingEdge 上升沿触发中断
- kGPIO_IT_FallingEdge 下降沿触发中断
- kGPIO_IT_RisingFallingEdge 上升和下降沿都触发中断
- kGPIO_IT_High 高电平触发中断

[in] **status** 是否使能

- 0 disable
- 1 enable

返回值

None

在文件 gpio.c 第 437 行定义.

```
void GPIO_PinConfig ( uint32_t              instance,
                      uint8_t               pin,
                      GPIO_PinConfig_Type   mode
                    )
```

设置引脚为输入还是输出功能 用户一般不必调用

注解

只有当引脚作为GPIO时才有意义

```
1 // 将PORTB端口的3引脚设置输入引脚
2 GPIO_PinConfig(HW_GPIOB, 3, kInpput);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

[in] **mode** 输入或者输出设置

- kInpput 输入功能选择
- kOutput 输出功能选择

返回值

None

在文件 gpio.c 第 164 行定义.

uint8_t GPIO_QuickInit (uint32_t instance,
uint32_t pinx,
GPIO_Mode_Type mode
)

快速初始化一个GPIO引脚 实际上是GPIO_Init的最简单配置

```
1 //初始化配置PORTB端口的10引脚为推挽输出引脚
2 GPIO_QuickInit(HW_GPIOB, 10, kGPIO_Mode_OPP);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pinx** 端口上的引脚号 0~31

[in] **mode** 引脚工作模式

- kGPIO_Mode_IFT 悬空输入
- kGPIO_Mode_IPD 下拉输入
- kGPIO_Mode_IPU 上拉输入
- kGPIO_Mode_OOD 开漏输出
- kGPIO_Mode_OPP 推挽输出

返回值

instance GPIO模块号

在文件 **gpio.c** 第 **243** 行定义.

uint8_t GPIO_ReadBit (uint32_t instance, uint8_t pin)

读取一个引脚上的电平状态

```
1 //读取PORTB端口的10引脚的电平状态
2 uint8_t status ; //用于存储引脚的状态
3 status = GPIO_ReadBit(HW_GPIOB, 10); //获取引脚的状态并存储到status
  中
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

返回值

0 低电平

1 高电平

在文件 **gpio.c** 第 **337** 行定义.

uint32_t GPIO_ReadPort (uint32_t instance)

读取一个端口32位的数据

```
1 //获取PORTB端口的所有引脚的电平状态
2 uint32_t status ; //用于存储引脚的状态
3 status = GPIO_ReadPort(HW_GPIOB); //获取引脚的状态并存储到status中
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

返回值

端口的32位数据

在文件 gpio.c 第 385 行定义.

void GPIO_ResetBit (uint32_t instance, uint32_t pin)

复位指定引脚的电平状态（即为低电平）

注解

此引脚首先配置成输出引脚

```
1 //复位PORTB端口的10引脚
2 GPIO_ResetBit(HW_GPIOB, 10);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

返回值

None

在文件 gpio.c 第 315 行定义.

void GPIO_SetBit (uint32_t instance, uint32_t pin)

置位指定引脚的电平状态（即为高电平）

注解

此引脚首先配置成输出引脚

```
1 //置位PORTB端口的10引脚
2 GPIO_SetBit(HW_GPIOB, 10);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

返回值

None

在文件 gpio.c 第 294 行定义.

void GPIO_ToggleBit (uint32_t instance, uint8_t pin)

翻转一个引脚的电平状态

```
1 //翻转PORTB端口的10引脚的电平状态
2 GPIO_ToggleBit(HW_GPIOB, 10);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

返回值

None

在文件 gpio.c 第 365 行定义.

void GPIO_WriteBit (uint32_t instance, uint8_t pin, uint8_t data)

设置指定引脚输出高电平或者低电平

注解

此引脚首先配置成输出引脚

```
1 //设置PORTB端口的10引脚输出高电平
2 GPIO_WriteBit(HW_GPIOB, 10, 1);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

[in] **data** 引脚的电平状态

- 0 低电平
- 1 高电平

返回值

None

在文件 gpio.c 第 272 行定义.

---

void GPIO_WritePort (uint32_t instance,
uint32_t data
)

向一个端口写入32位数据

```
1 //向PORTB端口写入0xFFFFFFFF
2 GPIO_WriteByte(HW_GPIOB, 0xFFFFFFFF);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **data** 32位数据

返回值

None

在文件 gpio.c 第 404 行定义.

---

void PORT_PinMuxConfig (uint32_t instance,
uint8_t pin,
PORT_PinMux_Type pinMux
)

set GPIO pin mux

注解

enable PORT clock before set pinmux number

参数

[in] **instance** GPIO模块号

- HW_GPIOx GPIOx moudle

[in] **pin** pin index number 0-31

[in] **pinMux** pinmux function

- kPinAltx Pinmux function x

返回值

None

在文件 gpio.c 第 58 行定义.

void PORT_PinOpenDrainConfig (uint32_t instance, uint8_t pin, bool status)

端口引脚的开漏状态配置 用户一般不必调用

```
1 // 将PORTA端口的3引脚设置为开漏状态
2 PORT_PinOpenDrainConfig(HW_GPIOA, 3, ENABLE);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

[in] **status** 功能开关控制

- ENABLE 开启功能
- DISABLE 关闭功能

返回值

None

在文件 gpio.c 第 115 行定义.

void PORT_PinPassiveFilterConfig (uint32_t instance, uint8_t pin, bool status)

端口引脚的开启无源滤波器 作为输入时有效

```
1 // 将PORTA端口的3引脚设置为开漏状态
2 PORT_PinPassiveFilterConfig(HW_GPIOA, 3, ENABLE);
```

参数

[in] **instance** GPIO模块号

- HW_GPIOA 芯片的PORTA端口
- HW_GPIOB 芯片的PORTB端口
- HW_GPIOC 芯片的PORTC端口
- HW_GPIOD 芯片的PORTD端口
- HW_GPIOE 芯片的PORTE端口

[in] **pin** 端口上的引脚号 0~31

[in] **status** 功能开关控制

- ENABLE 开启功能
- DISABLE 关闭功能

返回值

None

在文件 gpio.c 第 139 行定义.

---

void PORT_PinPullConfig (uint32_t instance,
uint8_t pin,
PORT_Pull_Type pull
)

set pin internal pullup/down resistors

注解

pull resistor value is about 20K

参数

[in] **instance** GPIO模块号

- HW_GPIOx GPIOx moudle

[in] **pin** pin index number 0-31

[in] **pull** pull select

- kPullDisabled disable pull resistor
- kPullUp pull up
- kPullDown pull down

返回值

None

在文件 gpio.c 第 77 行定义.

i2c.c 文件参考

浏览源代码.

函数

uint8_t	**I2C_QuickInit** (uint32_t MAP, uint32_t baudrate)
void	**I2C_Init** (I2C_InitTypeDef *I2C_InitStruct)
int	**I2C_BurstWrite** (uint32_t instance, uint8_t chipAddr, uint32_t addr, uint32_t addrLen, uint8_t *buf, uint32_t len)
int	**I2C_WriteSingleRegister** (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t data)
int	**I2C_BurstRead** (uint32_t instance, uint8_t chipAddr, uint32_t addr, uint32_t addrLen, uint8_t *buf, uint32_t len)
int	**I2C_Probe** (uint32_t instance, uint8_t chipAddr)
int	**I2C_ReadSingleRegister** (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t *data)
int	**SCCB_ReadSingleRegister** (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t *data)
int	**SCCB_WriteSingleRegister** (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t data)
void	**I2C_Scan** (uint32_t MAP)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.06 FreeXc 完善了对 i2c 模块的相关注释

注解
此文件为芯片IIC模块的底层功能函数

在文件 **i2c.c** 中定义.

函数说明

```
int I2C_BurstRead ( uint32_t  instance,
                    uint8_t   chipAddr,
                    uint32_t  addr,
                    uint32_t  addrLen,
                    uint8_t *  buf,
                    uint32_t  len
                  )
```

I2C read mutiple data.

参数

[in]	**instance**	instance of i2c moudle
[in]	**chipAddr**	i2c slave addr
[in]	**addr**	i2c slave register offset
[in]	**addrLen**	len of slave register addr(in byte)
[out]	**buf**	data buf
[in]	**len**	data length

返回值

0 success
1 failure

在文件 **i2c.c** 第 **309** 行定义.

---

int I2C_BurstWrite (uint32_t instance,
uint8_t chipAddr,
uint32_t addr,
uint32_t addrLen,
uint8_t * buf,
uint32_t len
)

I2C write mutiple data.

参数

[in]	**instance**	instance of i2c moudle
[in]	**chipAddr**	i2c slave addr
[in]	**addr**	i2c slave register offset
[in]	**addrLen**	len of slave register addr(in byte)
[in]	**buf**	data buf
[in]	**len**	data length

返回值

0 success
1 failure

在文件 **i2c.c** 第 **254** 行定义.

---

void I2C_Init (I2C_InitTypeDef * I2C_InitStruct)

I2C 初始化(待定义)

参数

[in] **I2C_InitStruct** 指向I2C结构体的指针

在文件 **i2c.c** 第 **100** 行定义.

---

int I2C_Probe (uint32_t instance,
uint8_t chipAddr
)

proble i2c bus

参数

[in] **instance** instance of i2c moudle
[in] **chipAddr** i2c slave addr

注解

see if it's available i2c slave on the bus

返回值

0 success
1 failure

在文件 **i2c.c** 第 **355** 行定义.

---

uint8_t I2C_QuickInit (uint32_t MAP,
uint32_t baudrate
)

I2C快速初始化函数

```
1 //PE00\PE01初始化为i2c功能，波特率为
2 uint32_t instance;
3 instance = I2C_QuickInit(I2C1_SCL_PE01_SDA_PE00,1000);
```

参数

[in] **MAP** I2C引脚配置缩略图,详见i2c.h
[in] **baudrate** 波特率(函数中暂未配置，取为默认)

返回值

i2c模块号

在文件 **i2c.c** 第 **41** 行定义.

---

int I2C_ReadSingleRegister (uint32_t instance,
uint8_t chipAddr,
uint8_t addr,
uint8_t * data
)

read single register value

参数

[in] **instance** instance of i2c moudle
[in] **chipAddr** i2c slave addr
[in] **addr** i2c slave register offset
[out] **data** data pointer

注解

usually used on i2c sensor devices

返回值

0 success
1 failure

在文件 **i2c.c** 第 **379** 行定义.

void I2C_Scan (uint32_t MAP)

i2c bus scan test

参数

[in] **MAP** I2C引脚配置缩略图,详见i2c.h

返回值

None

在文件 **i2c.c** 第 **465** 行定义.

int I2C_WriteSingleRegister (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t data)

write single register value

参数

[in] **instance** instance of i2c module
[in] **chipAddr** i2c slave addr
[in] **addr** i2c slave register offset
[in] **data** data to write

注解

usually used on i2c sensor devices

返回值

0 success
1 failure

在文件 **i2c.c** 第 **293** 行定义.

int SCCB_ReadSingleRegister (uint32_t instance, uint8_t chipAddr, uint8_t addr, uint8_t * data)

SCCB(protocol,the same as i2c) read single register value.

参数

[in] **instance** instance of i2c moudle
[in] **chipAddr** i2c slave addr

[in] **addr** i2c slave register offset
[out] **data** data pointer

注解
usually used on i2c sensor devices

返回值
0 success
1 failure

在文件 **i2c.c** 第 **394** 行定义.

---

int SCCB_WriteSingleRegister (uint32_t instance,
uint8_t chipAddr,
uint8_t addr,
uint8_t data
)

SCCB(protocol,the same as i2c) write single register value.

参数
[in] **instance** instance of i2c module
[in] **chipAddr** i2c slave addr
[in] **addr** i2c slave register offset
[in] **data** data to write

注解
usually used on i2c sensor devices

返回值
0 success
1 failure

在文件 **i2c.c** 第 **441** 行定义.

i2s.c 文件参考

浏览源代码.

函数

void	**SAI_HAL_SetMclkDiv** (uint32_t instance, uint32_t mclk, uint32_t src_clk)
void	**I2S_SetTxCmd** (uint32_t instance, bool val)
void	**I2S_TxSetProtocol** (uint32_t instance, I2S_Protocol_t protocol)
void	**I2S_SetSampleBit** (uint32_t instance, I2S_Protocol_t protocol, uint32_t bits)
void	**I2S_SetIntMode** (uint32_t instance, I2S_Int_t mode, bool val)
void	**I2S_TxSetSyncMode** (uint32_t instance, SAI_SyncMode_t mode)
void	**I2S_SendData** (uint32_t instance, uint32_t sampleBit, uint32_t chl, uint8_t *buf, uint32_t len)
void	**I2S_Init** (I2S_InitTypeDef *Init)

详细描述

作者
YANDLD

版本
V2.5

日期
2015.7.23
2015.10.08 FreeXc 完善了对 i2s 模块的相关注释

在文件 **i2s.c** 中定义.

函数说明

void I2S_Init (I2S_InitTypeDef * Init)

I2S 初始化

参数
[in] **Init** 指向I2S初始化结构体的指针

返回
None

在文件 **i2s.c** 第 **273** 行定义.

void I2S_SendData (uint32_t instance, uint32_t sampleBit, uint32_t chl, uint8_t * buf, uint32_t len

)

I2S Send Data.

参数

[in] **instance** I2S instance
[in] **sampleBit**
[in] **chl** 通道选择
[in] **buf** data pointer
[in] **len** data length

返回

None

在文件 **i2s.c** 第 **246** 行定义.

void I2S_SetIntMode (uint32_t instance,
I2S_Int_t mode,
bool val
)

I2S Interrupt Mode Config.

参数

[in] **instance** I2S instance
[in] **mode** 中断模式，详见i2s.h文件
[in] **val** enable or disable

- 0 disable
- 1 enable

返回

None

在文件 **i2s.c** 第 **190** 行定义.

void I2S_SetSampleBit (uint32_t instance,
I2S_Protocol_t protocol,
uint32_t bits
)

I2S Sample Bit Config.

参数

[in] **instance** I2S instance
[in] **protocol** 协议，详见i2s.h文件
[in] **bits** width

返回

None

在文件 **i2s.c** 第 **171** 行定义.

void I2S_SetTxCmd (uint32_t instance,
bool val
)

I2S Transmitter Enable and Bit Clock Enable.

参数

[in] **instance** I2S instance
[in] **val** enable or disable

- 0 disable
- 1 enable

返回

None

在文件 **i2s.c** 第 **83** 行定义.

void I2S_TxSetProtocol (uint32_t instance,
I2S_Protocol_t protocol
)

I2S Transmitter Protocol Config.

参数

[in] **instance** I2S instance
[in] **protocol** 协议，详见i2s.h文件

返回

None

在文件 **i2s.c** 第 **103** 行定义.

void I2S_TxSetSyncMode (uint32_t instance,
SAI_SyncMode_t mode
)

I2S Sync Mode config.

参数

[in] **instance** I2S instance
[in] **mode** 同步模式，详见i2s.h文件

返回

None

在文件 **i2s.c** 第 **215** 行定义.

void SAI_HAL_SetMclkDiv (uint32_t instance,

```
                                uint32_t  mclk,
                                uint32_t  src_clk
                               )
```

/brief internal function

在文件 **i2s.c** 第 **19** 行定义.

lptmr.c 文件参考

浏览源代码.

函数

void	**LPTMR_TC_Init** (LPTMR_TC_InitTypeDef *LPTMR_TC_InitStruct)
void	**LPTMR_PC_Init** (LPTMR_PC_InitTypeDef *LPTMR_PC_InitStruct)
void	**LPTMR_ITDMAConfig** (LPTMR_ITDMAConfig_Type config, bool status)
void	**LPTMR_CallbackInstall** (LPTMR_CallBackType AppCBFun)
uint32_t	**LPTMR_PC_ReadCounter** (void)
uint32_t	**LPTMR_PC_QuickInit** (uint32_t MAP)
void	**LPTMR_ClearCounter** (void)
void	**LPTimer_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.03 FreeXc 完善了lptmr模块的相关注释

在文件 **lptmr.c** 中定义.

函数说明

void LPTimer_IRQHandler (void)

中断处理函数入口，调用用户注册的回调函数

注解
函数内部用于中断事件处理

在文件 **lptmr.c** 第 **274** 行定义.

void LPTMR_CallbackInstall (LPTMR_CallBackType AppCBFun)

注册LPTMR中断回调函数

参数
[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 lptmr.c 第 188 行定义.

void LPTMR_ClearCounter (void)

清除脉冲计数器的脉冲数

```
1 //清除脉冲计数的个数
2 LPTMR_ClearCounter();
```

返回值

None

在文件 lptmr.c 第 263 行定义.

void LPTMR_ITDMAConfig (LPTMR_ITDMAConfig_Type config, bool status)

LPTM模块中断和DMA功能配置

```
1 //配置LPTM模块产生溢出中断
2 LPTMR_ITDMAConfig(kLPTMR_IT_TOF, true);
```

参数

[in] **config** LPTM中断类型

- kLPTMR_IT_Disable 关闭中断
- kLPTMR_IT_TOF 开启计数溢出中断

[in] **status** 是否使能中断

- 0 interrupt disable
- 1 interrupt enable

返回值

None

在文件 lptmr.c 第 163 行定义.

void LPTMR_PC_Init (LPTMR_PC_InitTypeDef * LPTMR_PC_InitStruct)

初始化配置LPTM模块处于脉冲计数模式

```
1 //设置LPTM工作在脉冲计数模式，计数上限是0xFFFE
2 LPTMR_PC_InitTypeDef LPTMR_PC_InitStruct1; //申请一个结构变量
3 LPTMR_PC_InitStruct1.timeInMs = 500;       //设置计时时间是500ms
4 LPTMR_TC_Init(&LPTMR_TC_InitStruct1);
```

参数

[in] **LPTMR_PC_InitStruct** LPTMR配置结构体

- counterOverflowValue 计数器计数上限，极限为0xFFFF
- inputSource 脉冲源选择
  kLPTMR_PC_InputSource_CMP0-CMP0作为脉冲计数时钟源 kLPTMR_PC_InputSource_ALT1-外部引脚LPTMR_ALT1作为外部计数时钟源
  kLPTMR_PC_InputSource_ALT2-外部引脚LPTMR_ALT2作为外部计数时钟源
- pinPolarity 脉冲计数极性选择
  kLPTMR_PC_PinPolarity_RigsingEdge 上升沿计数
  kLPTMR_PC_PinPolarity_FallingEdge 下降沿计数

返回值

None

在文件 **lptmr.c** 第 **92** 行定义.

uint32_t LPTMR_PC_QuickInit (uint32_t MAP)

LPTMR脉冲计数快速初始化

```
1 //LPTMR脉冲计数快速初始化
2 LPTMR_PC_QuickInit(LPTMR_ALT2_PC05);
```

参数

[in] **MAP** 单路脉冲计数模块

参见

MAP详细的宏定义请见lptmr.h文件

返回值

脉冲计数的模块号

在文件 **lptmr.c** 第 **227** 行定义.

uint32_t LPTMR_PC_ReadCounter (void)

获取脉冲计数器的脉冲数

```
1 //获取脉冲计数的个数
2 uint32_t counter;
3 counter = LPTMR_PC_ReadCounter();
```

返回值

脉冲计数个数

在文件 **lptmr.c** 第 **208** 行定义.

void LPTMR_TC_Init (LPTMR_TC_InitTypeDef * LPTMR_TC_InitStruct)

初始化配置LPTMR模块处于计时器模式

```
1 //设置LPTM工作在计时器模式，时间间隔是500毫秒
2 LPTMR_TC_InitTypeDef LPTMR_TC_InitStruct1; //申请一个结构变量
3 LPTMR_TC_InitStruct1.timeInMs = 500;        //设置计时时间是500ms
4 LPTMR_TC_Init(&LPTMR_TC_InitStruct1);
```

参数

[in] **LPTMR_TC_InitStruct** LPTMR配置结构体

- timeInMs 定时时间 单位为ms

返回值

None

在文件 **lptmr.c** 第 **50** 行定义.

nfc.c 文件参考

浏览源代码.

函数

uint32_t	**NFC_GetBufAddr** (uint8_t nfcbufNum)
void	**NFC_Init** (NFC_InitTypeDef *NFC_InitStruct)
void	**NFC_SendResetCmd** (void)
void	**NFC_ReadFlashID** (uint8_t nfcBufNum, uint32_t *id0, uint32_t *id1)
void	**NFC_BlockErase** (uint32_t cs, uint32_t row_addr)
void	**NFC_PageProgram** (uint8_t cs, uint8_t nfcbufNum, uint32_t row_addr, uint16_t col_addr)
void	**NFC_PageRead** (uint8_t cs, uint8_t nfcBufNum, uint32_t row_addr, uint16_t col_addr)

详细描述

作者
YANDLD

版本
V2.5

日期
2015.3.5
2015.10.04 FreeXc完善了nfc模块的注释

在文件 **nfc.c** 中定义.

函数说明

void NFC_BlockErase (uint32_t cs,
uint32_t row_addr
)

Erase NFC block.

Erases the NFC block containing the specified address. This function will only erase one block at a time. If multiple blocks need to be erased, then the function should be called once per block to erase.

参数

[in]	**cs**	chip select
[in]	**row_addr**	NAND flash row addr for the block to erase (up to 24 bits)

注解
raw_addr: how many page in this chip, there is no block addr meaning, so in blockerase fun:, raw_addr[0:5] is ignored coloum_addr: how many byte in a page,

在文件 **nfc.c** 第 **222** 行定义.

uint32_t NFC_GetBufAddr (uint8_t nfcbufNum)

get buffer address of NFC

参数

[in] **nfcbufNum** buffer num

返回

buffer address

在文件 **nfc.c** 第 **49** 行定义.

void NFC_Init (NFC_InitTypeDef * NFC_InitStruct)

initialize NFC struct

参数

[in] **NFC_InitStruct** 指向NFC初始化结构体指针

返回值

None

在文件 **nfc.c** 第 **59** 行定义.

void NFC_PageProgram (uint8_t cs, uint8_t nfcbufNum, uint32_t row_addr, uint16_t col_addr)

NFC page prpgram.

Programs a single page worth of data from the NFC into the NAND flash memory at the specified NFC address. This function will only program one NAND page at a time. If multiple pages need to be programmed, then the function should be called once per page to write. Data must be loaded into the NFC's buffers before calling this function.

参数

[in]	**cs**	chip select
[in]	**nfcbufNum**	first NFC internal buffer to program to the NAND (auto increment will be used)
[in]	**row_addr**	NAND flash row addr for the block to program (up to 24 bits)
[in]	**col_addr**	NAND flash col addr for the page to program (up to 16 bits)

注解

the column address should be aligned to the NAND page size

在文件 **nfc.c** 第 **283** 行定义.

void NFC_PageRead (uint8_t cs, uint8_t nfcBufNum,

```
                    uint32_t row_addr,
                    uint16_t col_addr
                  )
```

NFC page read.

Reads a single page worth of data from the NAND flash at the specified address into the NFC's buffers. This function will only read one NAND page at a time. If multiple pages need to be read, then the function should be called once per page to read. Data will be loaded into the NFC's buffers after the function completes.

参数

[in]	**cs**	chip select
[in]	**nfcBufNum**	first NFC internal buffer to program to the NAND (auto increment will be used)
[in]	**row_addr**	NAND flash row addr for the block to program (up to 24 bits)
[in]	**col_addr**	NAND flash col addr for the page to program (up to 16 bits)

注解

the column address should be aligned to the NAND page size

在文件 **nfc.c** 第 **342** 行定义.

```
void NFC_ReadFlashID ( uint8_t     nfcBufNum,
                       uint32_t *  id0,
                       uint32_t *  id1
                     )
```

read NFC flash

参数

[in]	**nfcBufNum**	first NFC internal buffer to program to the NAND (auto increment will be used)
[out]	**id0**	store the NFC flash ID0(SR1)
[out]	**id1**	store the NFC flash ID1(SR2)

返回值

None

在文件 **nfc.c** 第 **188** 行定义.

```
void NFC_SendResetCmd ( void  )
```

send NFC reset cmd

返回值

None

在文件 **nfc.c** 第 **164** 行定义.

pdb.c 文件参考

浏览源代码.

函数

void	**PDB_SoftwareTrigger** (void)
uint32_t	**PDB_GetMODValue** (void)
void	**PDB_QuickInit** (PDB_TriggerSrc_Type triggerSrc, uint32_t timeInUs)
void	**PDB_Init** (PDB_InitTypeDef *PDB_InitStruct)
void	**PDB_SetADCPreTrigger** (uint32_t adcInstance, uint32_t adcMux, uint32_t dlyValue, bool status)
void	**PDB_SetBackToBackMode** (uint32_t adcInstance, uint32_t adcMux, bool status)
void	**PDB_ITDMAConfig** (PDB_ITDMAConfig_Type config, bool status)
void	**PDB_CallbackInstall** (PDB_CallBackType AppCBFun)
void	**PDB0_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.24
2015.10.04 FreeXc完善了pdb模块的注释

在文件 **pdb.c** 中定义.

函数说明

void PDB0_IRQHandler (void)

PDB中断处理函数入口

PDB0_IRQHandler 芯片的PDB0模块中断函数入口

注解
该函数内部用于调用用户的中断处理函数,用户无需使用

在文件 **pdb.c** 第 **236** 行定义.

void PDB_CallbackInstall (PDB_CallBackType AppCBFun)

PDB注册中断回调函数

参数

[in] **AppCBFun** 回调函数指针入口

返回值

None

参见

对于此函数的具体应用请查阅应用实例

在文件 **pdb.c** 第 **221** 行定义.

uint32_t PDB_GetMODValue (void)

获得PDB Mod 寄存器

返回值

MOD寄存器值

在文件 **pdb.c** 第 **69** 行定义.

void PDB_Init (PDB_InitTypeDef * PDB_InitStruct)

PDB模块初始化

参数

[in] **PDB_InitStruct** 指向PDB初始化结构体的指针

参见

详细请参见PDB_QuickInit的函数定义

返回值

None

在文件 **pdb.c** 第 **104** 行定义.

void PDB_ITDMAConfig (PDB_ITDMAConfig_Type config, bool status)

PDB中断及DMA功能开关函数

参数

[in] **config** 中断及DMA配置

- kPDB_IT_CF 关闭中断
- kPDB_DMA_CF 关闭DMA功能

[in] **status** 是否使能中断或DMA

- 0 disable
- 1 enable

返回值

None

在文件 pdb.c 第 187 行定义.

---

void PDB_QuickInit (PDB_TriggerSrc_Type triggerSrc,
uint32_t timeInUs
)

PDB快速初始化

```
1 //开启PDB模块(软件触发)，定时10ms
2 PDB_QuickInit(kPDB_SoftwareTrigger, 10*1000);
```

参数

[in] **triggerSrc** PDB trigger source,详细请参见pdb.h文件
[in] **timeInUs** 定时时间，单位为微秒us

返回值

None

在文件 pdb.c 第 84 行定义.

---

void PDB_SetADCPreTrigger (uint32_t adcInstance,
uint32_t adcMux,
uint32_t dlyValue,
bool status
)

设置PDB触发ADC

参数

[in] **adcInstance** 需要触发的ADC模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **adcMux** ADC转换通道

- kADC_MuxA A通道模式
- kADC_MuxB B通道模式

[in] **dlyValue** 延时计数值(内部暂未使用)
[in] **status** 是否使能

- 1 enable
- 0 disable

返回值

None

在文件 pdb.c 第 143 行定义.

void PDB_SetBackToBackMode (uint32_t adcInstance, uint32_t adcMux, bool status)

PDB ADC pre-trigger operation as back-to-back mode.

参数

[in] **adcInstance** 需要触发的ADC模块号

- HW_ADC0 ADC0模块
- HW_ADC1 ADC1模块
- HW_ADC2 ADC2模块

[in] **adcMux** ADC转换通道

- kADC_MuxA A通道模式
- kADC_MuxB B通道模式

[in] **status** enable/disable the PDB ADC pre-trigger operation as back-to-back mode

- 0 disable
- 1 enable

参见

K60P144M100SF2RM PDB Chapter –> Channel n Control Register1

返回值

None

在文件 **pdb.c** 第 **170** 行定义.

void PDB_SoftwareTrigger (void)

软件触发PDB开始转换一次

返回值

None

在文件 **pdb.c** 第 **20** 行定义.

pit.c 文件参考

浏览源代码.

函数

void	**PIT_Init** (PIT_InitTypeDef *PIT_InitStruct)
void	**PIT_QuickInit** (uint8_t chl, uint32_t timeInUs)
void	**PIT_ITDMAConfig** (uint8_t chl, PIT_ITDMAConfig_Type config, bool flag)
void	**PIT_ResetCounter** (uint8_t chl)
uint32_t	**PIT_GetCounterValue** (uint8_t chl)
void	**PIT_CallbackInstall** (uint8_t chl, PIT_CallBackType AppCBFun)
void	**PIT0_IRQHandler** (void)
void	**PIT1_IRQHandler** (void)
void	**PIT2_IRQHandler** (void)
void	**PIT3_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.24
2015.10.03 FreeXc 完善了对 **pit.c** and pit.h文件的注释

注解
此文件为芯片PIT模块的底层功能函数

在文件 **pit.c** 中定义.

函数说明

void PIT_CallbackInstall (uint8_t chl, PIT_CallBackType AppCBFun)

注册中断回调函数

参数

[in] **chl** 通道号

- HW_PIT0_CH0 0通道入口
- HW_PIT0_CH1 1通道入口
- HW_PIT0_CH2 2通道入口

- HW_PIT0_CH3 3通道入口

[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 **pit.c** 第 **175** 行定义.

uint32_t PIT_GetCounterValue (uint8_t chl)

读取Counter值

参数

[in] **chl** 通道号

- HW_PIT0_CH0 0通道
- HW_PIT0_CH1 1通道
- HW_PIT0_CH2 2通道
- HW_PIT0_CH3 3通道

返回值

Counter值

在文件 **pit.c** 第 **158** 行定义.

void PIT_Init (PIT_InitTypeDef * PIT_InitStruct)

详细初始化PIT模块 推荐使用PIT_QuickInit函数

```
1 // 配置PIT0模块的0通道，时间周期为1ms
2 PIT_InitTypeDef PIT_InitStruct1; //申请一个结构变量
3 PIT_InitStruct1.chl = 0;          //选择0通道
4 PIT_InitStruct1.timeInUs = 1000  //1ms
5 PIT_Init(&PIT_InitStruct1);
```

参数

[in] **PIT_InitStruct** pit模块工作配置数据

返回值

None

在文件 **pit.c** 第 **39** 行定义.

void PIT_ITDMAConfig (uint8_t chl, PIT_ITDMAConfig_Type config, bool flag)

设置PIT模块是否开启中断功能

```
1  // 初始化PIT模块 0 通道 产生100MS中断 并开启中断 注册回调函数 在回调函数中打印调试信息
2  //声明中断回调函数
3  static void PIT0_CallBack(void);
4  //初始化PIT
5  PIT_QuickInit(HW_PIT_CH0, 100000);
6  PIT_CallbackInstall(HW_PIT0_CH0, PIT0_CallBack); //注册回调函数
7  PIT_ITDMAConfig(HW_PIT_CH0, kPIT_IT_TOF,ENABLE); //开启模块0通道中断
8  //中断回调函数编写
9  static void PIT0_CallBack(void)
10 {
11     printf("Enter PIT0 INt\r\n");
12 }
```

注意

在中断函数中尽量不要放置会导致阻塞的函数，比如printf等，上述实例程序仅作参考，建议实际应用中不加入printf函数

参数

[in] **chl** 通道号

- HW_PIT0_CH0 0通道
- HW_PIT0_CH1 1通道
- HW_PIT0_CH2 2通道
- HW_PIT0_CH3 3通道

[in] **config** 是否打开中断

- kPIT_IT_Disable 关闭中断
- kPIT_IT_TOF 定时器溢出中断

[in] **flag** 是否使能PIT中断

- ENABLE 使能
- DISABLE 不使能

返回值

None

在文件 pit.c 第 116 行定义.

void PIT_QuickInit (uint8_t chl,
uint32_t timeInUs
)

PIT模块快速初始化配置

```
1  // 初始化PIT模块 0 通道 产生100MS中断 并开启中断 注册回调函数 在回调函数中打印调试信息
2  //声明中断回调函数
3  static void PIT0_CallBack(void);
4  //初始化PIT模块的0通道，产生100ms中断
5  PIT_QuickInit(HW_PIT_CH0, 100000);
6  PIT_CallbackInstall(HW_PIT_CH0, PIT0_CallBack); //注册回调函数
7  PIT_ITDMAConfig(HW_PIT_CH0, kPIT_IT_TOF,ENABLE); //开启模块0通道中断
8  //中断回调函数
```

```
 9 static void PIT0_CallBack(void)
10 {
11     printf("Enter PIT0 INt\r\n");
12 }
```

注意

在中断函数中尽量不要放置会导致阻塞的函数，比如printf等，上述实例程序仅作参考，建议实际应用中不加入printf函数

参数

[in] **chl** 通道号

- HW_PIT_CH0
- HW_PIT_CH1
- HW_PIT_CH2
- HW_PIT_CH3

[in] **timeInUs** 产生中断的周期 \单位US

返回值

None

在文件 **pit.c** 第 **77** 行定义.

void PIT_ResetCounter (uint8_t chl)

PIT定时器Counter清到LOAD值 (重新开始新一次CountDown计时)

参数

[in] **chl** 通道号

- HW_PIT0_CH0 0通道
- HW_PIT0_CH1 1通道
- HW_PIT0_CH2 2通道
- HW_PIT0_CH3 3通道

返回值

None

在文件 **pit.c** 第 **144** 行定义.

rtc.c 文件参考

浏览源代码.

函数

int	**RTC_GetWeek** (int year, int month, int days)
void	**RTC_GetTime** (RTC_DateTime_Type *datetime)
bool	**RTC_IsTimeValid** (void)
void	**RTC_SetAlarm** (RTC_DateTime_Type *datetime)
void	**RTC_SetCompensation** (uint32_t compensationInterval, uint32_t timeCompensation)
void	**RTC_Init** (RTC_InitTypeDef *RTC_InitStruct)
void	**RTC_QuickInit** (void)
uint32_t	**RTC_GetTSR** (void)
uint32_t	**RTC_GetTAR** (void)
void	**RTC_SetTime** (RTC_DateTime_Type *datetime)
void	**RTC_SetTSR** (uint32_t val)
void	**RTC_ITDMAConfig** (RTC_ITDMAConfig_Type config, bool status)
void	**RTC_CallbackInstall** (RTC_CallBackType AppCBFun)
void	**RTC_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.03 FreeXc完善了rtc.c & rtc.h文件的相关注释

在文件 **rtc.c** 中定义.

函数说明

void RTC_CallbackInstall (RTC_CallBackType AppCBFun)

注册中断回调函数

参数
`[in]` **AppCBFun** 回调函数指针入口

返回值
None

注解

对于此函数的具体应用请查阅应用实例

在文件 rtc.c 第 391 行定义.

uint32_t RTC_GetTAR (void)

获得Time Alarm值

返回值

0 unvalid

!0 valid

在文件 rtc.c 第 306 行定义.

void RTC_GetTime (RTC_DateTime_Type * datetime)

获得RTC的时间

```
1 //获得RTC的时间
2 RTC_DateTime_Type ts;     //申请一个结构体
3 RTC_GetTime(&ts);     //将日期存储到ts中
```

参数

datetime 返回出来的年月日等信息结构体

返回值

None

在文件 rtc.c 第 156 行定义.

uint32_t RTC_GetTSR (void)

获得TSR值

返回值

0 unvalid

!0 valid

在文件 rtc.c 第 296 行定义.

int RTC_GetWeek (int year, int month, int days)

由年月日计算出周数

参数

[in] **year** 年

[in] **month** 月

[in] **days** 日

返回值

返回计算出来的周期数

在文件 rtc.c 第 39 行定义.

void RTC_Init (RTC_InitTypeDef * RTC_InitStruct)

RTC模块初始化配置，用来配置内部的电容参数

参数

[in] **RTC_InitStruct** 指向RTC初始化结构体指针

返回值

None

在文件 rtc.c 第 240 行定义.

void RTC_IRQHandler (void)

系统中断处理函数，调用用户定义的回调函数，此函数用户无需使用

注解

函数内部用于中断事件处理

在文件 rtc.c 第 402 行定义.

bool RTC_IsTimeValid (void)

判断当前RTC时钟模块时间是否有效

```
1   当时间无效（从来未执行过RTC时，初始化RTC的时间）
2 if(RTC_IsTimeValid())
3 {
4    printf("time invalid, reset time!\r\n");
5    RTC_SetTime(&td);
6 }
```

返回值

0 有效

!0 无效

在文件 rtc.c 第 179 行定义.

void RTC_ITDMAConfig (RTC_ITDMAConfig_Type config, bool status)

设置RTC中断功能

```
1 //设置RTC开启闹钟中断
2 RTC_ITDMAConfig(kRTC_IT_TimeAlarm, true);
```

参数

[in] **config** 配置中断类型

- kRTC_IT_TimeAlarm 闹钟中断
- kRTC_IT_TimeOverflow 时间溢出中断

[in] **status** 是否使能RTC中断

- 0 关闭中断
- 1 打开中断

返回值

None

在文件 **rtc.c** 第 **356** 行定义.

void RTC_QuickInit (void)

RTC模块快速初始化配置，设定内部电容为8pF.

返回值

None

在文件 **rtc.c** 第 **284** 行定义.

void RTC_SetAlarm (RTC_DateTime_Type * datetime)

设置闹钟时间

参数

[in] **datetime** 时间戳结构体

返回值

None

在文件 **rtc.c** 第 **193** 行定义.

void RTC_SetCompensation (uint32_t compensationInterval, uint32_t timeCompensation)

设置RTC补偿寄存器

参数

compensationInterval Configures the compensation interval in seconds from 1 to 256 to control how frequently the TCR should adjust the number of 32.768 kHz cycles in each second. The value written should be one less than the number of seconds (for example, write zero to configure for a compensation interval of one second). This register is double buffered and writes do not take affect until the end of the current compensation interval.

timeCompensation	Configures the number of 32.768 kHz clock cycles in each second. This register is double buffered and writes do not take affect until the end of the current compensation interval. 80h Time prescaler register overflows every 32896 clock cycles. ... ... FFh Time prescaler register overflows every 32769 clock cycles. 00h Time prescaler register overflows every 32768 clock cycles. 01h Time prescaler register overflows every 32767 clock cycles. ... ... 7Fh Time prescaler register overflows every 32641 clock cycles.

返回值

None

在文件 **rtc.c** 第 **227** 行定义.

void RTC_SetTime (RTC_DateTime_Type * datetime)

设置RTC的时间

参数

[in] **datetime** 指向时间的结构体指针

返回值

None

在文件 **rtc.c** 第 **316** 行定义.

void RTC_SetTSR (uint32_t val)

设置RTC的Time Seconds Register

参数

[in] **val** time second vlaue

返回值

None

在文件 **rtc.c** 第 **335** 行定义.

sd.c 文件参考

浏览源代码.

函数

uint32_t	**SD_StatusWait** (uint32_t mask)
uint32_t	**SDHC_SendCmd** (SDHC_Cmd_t *cmd)
uint32_t	**SD_QuickInit** (uint32_t baudrate)
uint8_t	**SD_Init** (SD_InitTypeDef *Init)
uint8_t	**SDHC_ReadBlock** (uint32_t sector, uint8_t *buf, uint32_t len)
uint8_t	**SDHC_WriteBlock** (uint32_t sector, uint8_t *buf, uint32_t len)
uint8_t	**SD_ReadSingleBlock** (uint32_t sector, uint8_t *buf)
uint8_t	**SD_WriteSingleBlock** (uint32_t sector, uint8_t *buf)
uint32_t	**SD_GetSizeInMB** (void)
uint8_t	**SD_ReadMultiBlock** (uint32_t sector, uint8_t *buf, uint32_t len)
uint8_t	**SD_WriteMultiBlock** (uint32_t sector, uint8_t *buf, uint32_t len)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.24
2015.10.04 FreeXc 完善了 sd 模块的相关注释

在文件 **sd.c** 中定义.

函数说明

uint32_t SD_GetSizeInMB (void)

GetSD size.

返回值
size in MB

在文件 **sd.c** 第 **628** 行定义.

uint8_t SD_Init (SD_InitTypeDef * Init)

SDHC complete initialize.

参数

[in] **Init** SD初始化结构体指针

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **429** 行定义.

uint32_t SD_QuickInit (uint32_t baudrate)

SDHC quick initialize.

参数

[in] **baudrate** 波特率

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **399** 行定义.

uint8_t SD_ReadMultiBlock (uint32_t sector, uint8_t * buf, uint32_t len)

SD_ReadMultiBlock legcy support.

注解

this function is same as SDHC_ReadBlock(...)

在文件 **sd.c** 第 **683** 行定义.

uint8_t SD_ReadSingleBlock (uint32_t sector, uint8_t * buf)

read SD single block data

参数

[in] **sector** 块

[out] **buf** 数据的存放地址

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **607** 行定义.

uint32_t SD_StatusWait (uint32_t mask)

等待状态位

参数

[in] **mask** 相关标志位

返回

相对应的状态

在文件 **sd.c** 第 **662** 行定义.

```
uint8_t SD_WriteMultiBlock ( uint32_t  sector,
                             uint8_t *  buf,
                             uint32_t  len
                           )
```

SD_WriteMultiBlock legcy support.

注解

this function is same as SDHC_WriteBlock(...)

在文件 **sd.c** 第 **692** 行定义.

```
uint8_t SD_WriteSingleBlock ( uint32_t  sector,
                              uint8_t *  buf
                            )
```

write SD single block data

参数

[in] **sector** 块

[in] **buf** 待写入数据的地址

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **619** 行定义.

```
uint8_t SDHC_ReadBlock ( uint32_t  sector,
                         uint8_t *  buf,
                         uint32_t  len
                       )
```

SDHC 块读操作

参数

[in] **sector** 块

[out] **buf** 数据的存放地址

[in] **len** 个数

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **470** 行定义.

uint32_t SDHC_SendCmd (SDHC_Cmd_t * cmd)

Set SDHC baud rate.

参数

[in] **cmd** 指令指针

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **191** 行定义.

uint8_t SDHC_WriteBlock (uint32_t sector, uint8_t * buf, uint32_t len)

SDHC 块写操作

参数

[in] **sector** 块

[in] **buf** 待写入数据的地址

[in] **len** 个数

返回值

0 ESDHC_OK

other error code

在文件 **sd.c** 第 **527** 行定义.

spi.c 文件参考

浏览源代码.

函数

void	**SPI_Init** (SPI_InitTypeDef *SPI_InitStruct)
void	**SPI_CTARConfig** (uint32_t instance, uint32_t ctar, SPI_FrameFormat_Type frameFormat, uint8_t dataSize, uint8_t bitOrder, uint32_t baudrate)
uint32_t	**SPI_QuickInit** (uint32_t MAP, SPI_FrameFormat_Type frameFormat, uint32_t baudrate)
void	**SPI_EnableTxFIFO** (uint32_t instance, bool status)
void	**SPI_EnableRxFIFO** (uint32_t instance, bool status)
void	**SPI_ITDMAConfig** (uint32_t instance, SPI_ITDMAConfig_Type config, bool status)
void	**SPI_CallbackInstall** (uint32_t instance, SPI_CallBackType AppCBFun)
uint16_t	**SPI_ReadWriteByte** (uint32_t instance, uint32_t ctar, uint16_t data, uint16_t CSn, SPI_PCS_Type csState)
void	**SPI0_IRQHandler** (void)
void	**SPI1_IRQHandler** (void)
void	**SPI2_IRQHandler** (void)

变量

SPI_Type *const	**SPI_InstanceTable** [] = SPI_BASES

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.07 FreeXc 完善了对 spi 模块的相关注释

注解
此文件为芯片SPI模块的底层功能函数

在文件 **spi.c** 中定义.

函数说明

void SPI_CallbackInstall (uint32_t instance,
SPI_CallBackType AppCBFun
)

注册中断回调函数

参数

[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 spi.c 第 429 行定义.

```
void SPI_CTARConfig ( uint32_t                instance,
                      uint32_t                ctar,
                      SPI_FrameFormat_Type    frameFormat,
                      uint8_t                 dataSize,
                      uint8_t                 bitOrder,
                      uint32_t                baudrate
                    )
```

SPI 波特率及传输控制寄存器配置

参数

[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **ctar** SPI通信通道选择

- HW_CTAR0 0配置寄存器
- HW_CTAR1 1配置寄存器

[in] **frameFormat** SPI通信时的相位和极性的关系

- kSPI_CPOL0_CPHA0
- kSPI_CPOL1_CPHA0
- kSPI_CPOL0_CPHA1
- kSPI_CPOL1_CPHA1

[in] **dataSize** 数据大小

[in] **bitOrder** LSB First

- 0 Data is transferred MSB first
- 1 Data is transferred LSB first

[in] **baudrate** SPI通信速度设置

返回

None

在文件 spi.c 第 228 行定义.

void SPI_EnableRxFIFO (uint32_t instance, bool status)

使能SPI接收的FIFO功能

参数

[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **status** enable or disable Tx FIFO

- 0 disable
- 1 enable

返回值

None

在文件 spi.c 第 353 行定义.

void SPI_EnableTxFIFO (uint32_t instance, bool status)

使能SPI发送的FIFO功能

参数

[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **status** enable or disable Tx FIFO

- 0 disable
- 1 enable

返回值

None

在文件 spi.c 第 332 行定义.

void SPI_Init (SPI_InitTypeDef * SPI_InitStruct)

初始化SPI模块

注解
需要其它函数配合使用，具体可参考SPI_QuickInit内部定义

参数
[in] **SPI_InitStruct** 指向SPI初始化配置结构体的指针

返回
None

在文件 spi.c 第 157 行定义.

void SPI_ITDMAConfig (uint32_t instance, SPI_ITDMAConfig_Type config, bool status)

SPI模块 中断和DMA功能配置

```
1 //使用SPI的1模块发送完成中断
2 SPI_ITDMAConfig(HW_SPI1, kSPI_IT_TCF, true);
```

参数
[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **config** SPI中断类型

- kSPI_IT_TCF 开启发送完成中断
- kSPI_DMA_TFFF TxFIFO 空 DMA请求
- kSPI_DMA_RFDF RxFIFO 空 DMA请求

[in] **status** enable or disable IT/DMA

- 0 disable
- 1 enable

返回值
None

在文件 spi.c 第 384 行定义.

uint32_t SPI_QuickInit (uint32_t MAP, SPI_FrameFormat_Type frameFormat, uint32_t baudrate)

快速初始化SPI模块

```
1 //使用SPI的1模块SCK-PE02 SOUT-PE01 SIN-PE03 通信速度为48000hz 极性和
```

```
    相位都是0
2 SPI_QuickInit(SPI1_SCK_PE02_SOUT_PE01_SIN_PE03,
  kSPI_CPOL0_CPHA0, 48000);
```

参数

[in] **MAP** SPI通信快速配置引脚预定义，详见spi.h文件

[in] **frameFormat** SPI通信时的相位和极性的关系

- kSPI_CPOL0_CPHA0
- kSPI_CPOL1_CPHA0
- kSPI_CPOL0_CPHA1
- kSPI_CPOL1_CPHA1

[in] **baudrate** SPI通信速度设置

返回

SPI模块号

在文件 **spi.c** 第 **299** 行定义.

---

uint16_t SPI_ReadWriteByte (uint32_t instance,
uint32_t ctar,
uint16_t data,
uint16_t CSn,
SPI_PCS_Type csState
)

SPI读写一字节数据

```
1  //使用SPI的1模块的1片选信号写一字节的数据0x55，片选信号最后为选中状态
2 SPI_ReadWriteByte(HW_SPI1, HW_CTAR0, 0x55, 1,
  kSPI_PCS_ReturnInactive);
```

参数

[in] **instance** 芯片SPI端口

- HW_SPI0 芯片的SPI0端口
- HW_SPI1 芯片的SPI1端口
- HW_SPI2 芯片的SPI2端口

[in] **ctar** SPI通信通道选择

- HW_CTAR0 0配置寄存器
- HW_CTAR1 1配置寄存器

[in] **data** 要发送的一字节数据

[in] **CSn** 片选信号端口选择

[in] **csState** 片选信号最后的状态

- kSPI_PCS_ReturnInactive 最后处于选中状态
- kSPI_PCS_KeepAsserted 最后保持未选中状态

返回

读取到的数据

在文件 **spi.c** 第 **457** 行定义.

systick.c 文件参考

浏览源代码.

函数

void	**SYSTICK_Init** (uint32_t timeInUs)
void	**SYSTICK_DelayInit** (void)
void	**SYSTICK_Cmd** (bool NewState)
void	**SYSTICK_ITConfig** (bool NewState)
void	**SYSTICK_DelayUs** (uint32_t us)
void	**SYSTICK_DelayMs** (uint32_t ms)
uint32_t	**SYSTICK_GetVal** (void)

详细描述

作者

YANDLD

版本

V2.5

日期

2014.3.26

2015.10.03 FreeXc完善了systick模块的相关注释

注解

此文件为芯片ARM内核中的SysTick模块的底层功能函数

在文件 **systick.c** 中定义.

函数说明

void SYSTICK_Cmd (bool NewState)

开启或者停止SysTick时钟

```
1 //开启时钟
2 SYSTICK_Cmd(true);
```

参数

[in] **NewState** 使能或者关闭

- true 使能
- false 停止

注意

当给微控制器移植OS后，需要开启systick时钟以及中断，不然OS创建的任务无法工作

返回值

None

在文件 systick.c 第 69 行定义.

void SYSTICK_DelayInit (void)

初始化SysTick为延时应用 初始化后就可以调用 DelayMs DelayUs

```
1 //将SysTick用作延时的初始化  初始化后系统延时20ms
2 SYSTICK_DelayInit();
3 SYSTICK_DelayMs(20);
```

返回值

None

在文件 systick.c 第 50 行定义.

void SYSTICK_DelayMs (uint32_t ms)

毫秒级延时函数

```
1 //延时100ms
2 SYSTICK_DelayMs(100);
```

参数

[in] **ms** 延时

返回值

None

在文件 systick.c 第 123 行定义.

void SYSTICK_DelayUs (uint32_t us)

微秒级延时函数

```
1 //延时100us
2 SYSTICK_DelayUs(100);
```

参数

[in] **us** 延时时间 单位us

返回值

None

在文件 systick.c 第 100 行定义.

uint32_t SYSTICK_GetVal (void)

获得当前System tick timer的值

返回值

当前System tick timer的值

在文件 systick.c 第 144 行定义.

void SYSTICK_Init (uint32_t timeInUs)

初始化SysTick时钟

```
1 // 初始化SysTick时钟 设定中断周期为10000us(10ms)
2 SYSTICK_Init(10000);
```

参数

[in] **timeInUs** 中断周期,单位us

注解

systick属于cm4内核中的模块，在RTOS中可作为其时钟节拍

返回值

None

在文件 systick.c 第 30 行定义.

void SYSTICK_ITConfig (bool NewState)

开启SysTick中断

```
1 //开启中断功能
2 SYSTICK_ITConfig(true);
```

参数

[in] **NewState** 使能或者关闭

- true 使能
- false 禁止

注意

当给微控制器移植OS后，需要开启systick时钟以及中断，不然OS创建的任务无法工作

返回值

None

在文件 systick.c 第 86 行定义.

tsi.c 文件参考

浏览源代码.

函数

uint32_t	**TSI_GetCounter** (uint32_t chl)
void	**TSI_Init** (TSI_InitTypeDef *TSI_InitStruct)
uint32_t	**TSI_QuickInit** (uint32_t MAP)
void	**TSI_ITDMAConfig** (TSI_ITDMAConfig_Type config)
void	**TSI_CallbackInstall** (TSI_CallBackType AppCBFun)
void	**TSI0_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.08 FreeXc 完善了对 tsi 模块的相关注释

注解
此文件为芯片TSI模块的底层功能函数

在文件 **tsi.c** 中定义.

函数说明

void TSI0_IRQHandler (void)

中断处理函数入口

注解
函数内部用于中断事件处理(调用用户注册的回调函数)

在文件 **tsi.c** 第 **209** 行定义.

void TSI_CallbackInstall (TSI_CallBackType AppCBFun)

注册中断回调函数

参数
[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 tsi.c 第 198 行定义.

uint32_t TSI_GetCounter (uint32_t chl)

获得指定通道的数值

参数

[in] **chl** tsi模块的通道1~15，详细请见tsi.h文件

返回值

该通道的数据值

在文件 tsi.c 第 46 行定义.

void TSI_Init (TSI_InitTypeDef * TSI_InitStruct)

TSI初始化配置

```
1 //初始化配置TSI模块的1通道为周期触发方式，判断阀值为200
2 TSI_InitTypeDef  TSI_InitStruct1;       //申请一个结构变量
3 TSI_InitStruct1.chl = 1;    //选择通道
4 TSI_InitStruct1.triggerMode = kTSI_TriggerPeriodicalScan; //硬件
  周期扫描
5 TSI_InitStruct1.threshld = 200;    //设置阀值为200
6 TSI_Init(&TSI_InitStruct1);
```

参数

[in] **TSI_InitStruct** 指向TSI初始化结构体的指针,存储通道的工作状态

返回值

None

在文件 tsi.c 第 84 行定义.

void TSI_ITDMAConfig (TSI_ITDMAConfig_Type config)

TSI模块中断类型或者DMA功能设置

```
1 //开启扫描结束触发中断模式
2 TSI_ITDMAConfig(kTSI_IT_EndOfScan);
```

参数

[in] **config** 中断配置模式

- kTSI_IT_Disable 关闭中断功能
- kTSI_IT_OutOfRange 超出阀值触发中断
- kTSI_IT_EndOfScan 扫描结束触发中断

返回值

None

在文件 **tsi.c** 第 **170** 行定义.

uint32_t TSI_QuickInit (uint32_t MAP)

TSI快速初始化配置

```
1 //快速初始化配置TSI模块的1通道的PTA0引脚做默认触控引脚
2 TSI_QuickInit(TSI0_CH1_PA00);
```

参数

[in] **MAP** TSI初始化预定义，详见tsi.h文件

返回值

通道号

在文件 **tsi.c** 第 **136** 行定义.

uart.c 文件参考

浏览源代码.

函数

__weak int	**fputc** (int ch, FILE *f)
__weak int	**fgetc** (FILE *f)
__weak size_t	**__write** (int handle, const unsigned char *buffer, size_t size)
__weak size_t	**__read** (int handle, unsigned char *buffer, size_t size)
int	**UART_printf** (uint32_t instance, const char *fmt,...)
void	**UART_Init** (UART_InitTypeDef *Init)
void	**UART_DeInit** (uint32_t instance)
void	**UART_SelectDebugInstance** (uint32_t instance)
void	**UART_EnableTxFIFO** (uint32_t instance, bool status)
void	**UART_EnableRxFIFO** (uint32_t instance, bool status)
uint32_t	**UART_GetTxFIFOSize** (uint32_t instance)
uint32_t	**UART_GetRxFIFOSize** (uint32_t instance)
void	**UART_SetTxFIFOWatermark** (uint32_t instance, uint32_t size)
void	**UART_SetRxFIFOWatermark** (uint32_t instance, uint32_t size)
void	**UART_WriteByte** (uint32_t instance, uint16_t ch)
uint8_t	**UART_ReadByte** (uint32_t instance, uint16_t *ch)
void	**UART_ITDMAConfig** (uint32_t instance, UART_ITDMAConfig_Type config, bool status)
void	**UART_CallbackTxInstall** (uint32_t instance, UART_CallBackTxType AppCBFun)
void	**UART_CallbackRxInstall** (uint32_t instance, UART_CallBackRxType AppCBFun)
uint8_t	**UART_QuickInit** (uint32_t MAP, uint32_t baudrate)
void	**UART0_RX_TX_IRQHandler** (void)
void	**UART1_RX_TX_IRQHandler** (void)
void	**UART2_RX_TX_IRQHandler** (void)
void	**UART3_RX_TX_IRQHandler** (void)
void	**UART4_RX_TX_IRQHandler** (void)
void	**UART5_RX_TX_IRQHandler** (void)
void	**UART_SetDMATxMode** (uint32_t instance, bool status)
void	**UART_DMASendByte** (uint32_t instance, uint8_t *buf, uint32_t size)
uint32_t	**UART_DMAGetRemainByte** (uint32_t instance)

变量

	C
FILE	**__stdout**
FILE	**__stdin**

FILE __stdin

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.25
2015.10.06 FreeXc 完善了对 uart 模块的相关注释

在文件 **uart.c** 中定义.

函数说明

```
__weak int fputc ( int      ch,
                   FILE *  f
                 )
```

put char, called by printf

注解
若串口初始化完成之后，调用printf函数会重定向到uart的发送函数
即需要打印的数据可以通过PC的串口调试助手(终端)打印出来

在文件 **uart.c** 第 **95** 行定义.

```
void UART_CallbackRxInstall ( uint32_t                instance,
                              UART_CallBackRxType  AppCBFun
                            )
```

注册接收中断回调函数

参数
[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意
具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数
[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 uart.c 第 685 行定义.

void UART_CallbackTxInstall (uint32_t instance, UART_CallBackTxType AppCBFun)

注册发送中断回调函数

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应用请查阅应用实例

在文件 uart.c 第 661 行定义.

void UART_DeInit (uint32_t instance)

Uart Deinitialization.

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

返回值

None

在文件 uart.c 第 359 行定义.

uint32_t UART_DMAGetRemainByte (uint32_t instance)

UART在DMA模式下，获得DMA主循环的次数

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

返回值

DMA主循环的次数

在文件 uart.c 第 978 行定义.

void UART_DMASendByte (uint32_t instance, uint8_t * buf, uint32_t size)

UART在DMA模式下发送数据

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **buf** 指向需要发送数据的指针

[in] **size** 发送数据的个数

返回值

None

在文件 uart.c 第 957 行定义.

void UART_EnableRxFIFO (uint32_t instance, bool status)

使能UART接收的FIFO功能

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **status** enable or disable Rx FIFO

- 0 disable
- 1 enable

返回值

None

在文件 uart.c 第 421 行定义.

void UART_EnableTxFIFO (uint32_t instance, bool status)

使能UART发送的FIFO功能

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **status** enable or disable Tx FIFO

- 0 disable
- 1 enable

返回值

None

在文件 **uart.c** 第 **395** 行定义.

uint32_t UART_GetRxFIFOSize (uint32_t instance)

获取UART接收中FIFO的通道深度

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

返回

Receive FIFO/Buffer Depth

在文件 **uart.c** 第 **458** 行定义.

uint32_t UART_GetTxFIFOSize (uint32_t instance)

获取UART发送中FIFO的通道深度

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

返回

Transmit FIFO/Buffer Depth

在文件 uart.c 第 440 行定义.

void UART_Init (UART_InitTypeDef * Init)

初始化UART模块

注解

用户需自己进行引脚的复用配置

```
1 //使用UART0模块 使用115200波特率进行通信
2 UART_InitTypeDef UART_InitStruct1;       //申请一个结构变量
3 UART_InitStruct1.instance = HW_UART0;    //选择UART0模块
4 UART_InitStruct1.baudrate = 115200;      //设置通信速度为115200
5 UART_InitStruct1.parityMode = kUART_ParityDisabled; //校验位
  禁止
6 UART_InitStruct1.bitPerChar = kUART_8BitsPerChar;    //每
  帧8bit
7 UART_Init(&UART_InitStruct1);
```

参数

[in] **Init** 指向串口工作配置存储结构体的指针,详细的定义请见uart.h文件

返回值

None

在文件 uart.c 第 248 行定义.

void UART_ITDMAConfig (uint32_t instance, UART_ITDMAConfig_Type config, bool status)

配置UART模块的中断或DMA属性

```
1 //配置UART0模块开启接收中断功能
2 UART_ITDMAConfig(HW_UART0, kUART_IT_Rx, true);
```

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **status** 使能开关

- 0 disable

- 1 enable

[in] **config** 工作模式选择

- kUART_IT_Tx
- kUART_DMA_Tx
- kUART_IT_Rx
- kUART_DMA_Rx

返回值

None

在文件 **uart.c** 第 **603** 行定义.

---

uint8_t UART_QuickInit (uint32_t MAP,
uint32_t baudrate
)

串口快速化配置函数

```
1 // 初始化 UART4 属性: 115200-N-8-N-1, Tx:PC15 Rx:PC14
2 UART_QuickInit(UART4_RX_PC14_TX_PC15, 115200);
```

参数

[in] **MAP** 串口引脚配置缩略图,详见uart.h

注解

例如 UART1_RX_PE01_TX_PE00，使用串口1的PTE1/PTE0引脚

参数

[in] **baudrate** 波特率 9600 115200...

返回值

UART模块号

在文件 **uart.c** 第 **707** 行定义.

---

uint8_t UART_ReadByte (uint32_t instance,
uint16_t * ch
)

UART接受一个字节

注解

非阻塞式接收 立即返回

```
1 //接收UART0模块的数据
2 uint8_t data; //申请变量，存储接收的数据
3 UART_ReadByte(HW_UART0, &data);
```

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口

- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意
具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数
[in] **ch** 接收到的数据指针

返回值
0 成功接收到数据
非0 没有接收到数据

在文件 **uart.c** 第 **565** 行定义.

---

void UART_SetDMATxMode (uint32_t instance,
bool status
)

设置UART为DMA发送模式

参数
[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意
具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数
[in] **status** 使能开关

- 0 disable
- 1 enable

返回值
None

在文件 **uart.c** 第 **904** 行定义.

---

void UART_SetRxFIFOWatermark (uint32_t instance,
uint32_t size
)

设定接收FIFO通道的水位

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **size** Receive Watermark Size

返回值

None

在文件 **uart.c** 第 **498** 行定义.

void UART_SetTxFIFOWatermark (uint32_t instance, uint32_t size)

设定发送FIFO通道的水位

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **size** Transmit Watermark Size

返回值

None

在文件 **uart.c** 第 **477** 行定义.

void UART_WriteByte (uint32_t instance, uint16_t ch)

串口发送一个字节

注意

阻塞式发送 只有发送完后才会返回

```
1 //使用UART0模块 发送数据0x5A
2 UART_WriteByte(HW_UART0, 0x5A);
```

参数

[in] **instance** 芯片串口端口

- HW_UART0 芯片的UART0端口
- HW_UART1 芯片的UART1端口
- HW_UART2 芯片的UART2端口
- HW_UART3 芯片的UART3端口
- HW_UART4 芯片的UART4端口
- HW_UART5 芯片的UART5端口

注意

具体的UART资源依芯片而不同，请参见相应的引脚复用说明

参数

[in] **ch** 需要发送的一字节数据

返回值

None

在文件 **uart.c** 第 **523** 行定义.

变量说明

C

初始值:

```
{
#endif


#ifdef __CC_ARM
struct __FILE
{
    int handle;



}
```

vref.c 文件参考

浏览源代码.

函数

void	**VREF_Init** (VREF_InitTypeDef *VREF_InitStruct)
void	**VREF_QuickInit** (void)
void	**VREF_DeInit** (void)
void	**VREF_SetTrimValue** (uint32_t val)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.26
2015.10.03 FreeXc完善了vref模块的相关注释

在文件 **vref.c** 中定义.

函数说明

void VREF_DeInit (void)

恢复VREF到默认状态

返回值
None

在文件 **vref.c** 第 **51** 行定义.

void VREF_Init (VREF_InitTypeDef * VREF_InitStruct)

初始化VREF 模块

参数
[in] **VREF_InitStruct** 指向VREF初始化结构体的指针

返回值
None

在文件 **vref.c** 第 **20** 行定义.

void VREF_QuickInit (void)

快速初始化VREF模块

注解

VREF被初始化后 需要经过大概35MS 才能有稳定的参考电压输出

注意

bufferMode对于不同的芯片，其可选的参数视不同的，比如K60只能是00和10，而K64有三种模式可选

返回值

None

在文件 **vref.c** 第 **39** 行定义.

void VREF_SetTrimValue (uint32_t val)

设置VREF校准值

注意

the trim value changes the resulting VREF by approximately ± 0.5 mV for each step

It is ± 0.5 mV,pay attention to the unit

参数

`[in]` **val** 校准值

返回值

None

在文件 **vref.c** 第 **64** 行定义.

wdog.c 文件参考

浏览源代码.

函数

void	**WDOG_QuickInit** (uint32_t timeInMs)
void	**WDOG_Init** (WDOG_InitTypeDef *WDOG_InitStruct)
void	**WDOG_ITDMAConfig** (bool status)
void	**WDOG_CallbackInstall** (WDOG_CallBackType AppCBFun)
uint32_t	**WDOG_GetResetCounter** (void)
void	**WDOG_ClearResetCounter** (void)
uint32_t	**WDOG_GetCurrentCounter** (void)
void	**WDOG_Refresh** (void)
void	**Watchdog_IRQHandler** (void)

详细描述

作者
YANDLD

版本
V2.5

日期
2014.3.24
2015.10.05 FreeXc 完善了wdog模块的相关API注释

注解
此文件为芯片看门狗模块的底层功能函数

在文件 **wdog.c** 中定义.

函数说明

void Watchdog_IRQHandler (void)

中断处理函数入口

注解
用于调用用户注册的回调函数,用户无需使用

在文件 **wdog.c** 第 **191** 行定义.

void WDOG_CallbackInstall (WDOG_CallBackType AppCBFun)

WDOG注册中断回调函数

参数

[in] **AppCBFun** 回调函数指针入口

返回值

None

注解

对于此函数的具体应该请查阅应用实例

在文件 **wdog.c** 第 **119** 行定义.

void WDOG_ClearResetCounter (void)

清除看门狗计数器的数值

```
1 //清除当前看门狗中计时器的数值
2 WDOG_ClearResetCounter();
```

返回值

None

在文件 **wdog.c** 第 **149** 行定义.

uint32_t WDOG_GetCurrentCounter (void)

读取看门狗当前定时器的输出值

返回

Watchdog Timer Output Value

在文件 **wdog.c** 第 **158** 行定义.

uint32_t WDOG_GetResetCounter (void)

读取看门狗计数器的数值

```
1 //获取当前看门狗中计时器的数值
2 uint32_t counter;     //申请一个变量
3 counter = WDOG_GetResetCounter();   //获取计时器的数值，存储
  在counter中
```

返回值

当前计数器的数值

在文件 **wdog.c** 第 **136** 行定义.

void WDOG_Init (WDOG_InitTypeDef * WDOG_InitStruct)

看门狗详细初始化配置

```
1   //配置看门狗在正常模式下，时间为100ms
2 WDOG_InitTypeDef WDOG_InitStruct1;          //申请一个结构变量
3 WDOG_InitStruct1.mode = kWDOG_Mode_Normal;//选择看门狗处于正常模式
```

```
4 WDOG_InitStruct1.timeOutInMs = 100;          //设置间隔时间为100ms
5 WDOG_InitStruct1.windowInMs  = 20;           //在正常模式下无意义
6 WDOG_Init(&WDOG_InitStruct1);
```

参数

[in] **WDOG_InitStruct** 看门狗工作模式配置结构体

返回值

None

在文件 **wdog.c** 第 **58** 行定义.

void WDOG_ITDMAConfig (bool status)

看门狗中断配置

```
1 //开启看门狗中断功能
2 WDOG_ITDMAConfig(true);   //中断不常用
```

参数

[in] **status** 是否开启WDOG中断

- true 开启中断
- false 关闭中断

返回值

None

在文件 **wdog.c** 第 **101** 行定义.

void WDOG_QuickInit (uint32_t timeInMs)

看门狗快速初始化配置

```
1 //配置看门狗的定时时间为100ms
2 WDOG_QuickInit(100);
```

参数

[in] **timeInMs** 看门狗触发时间，单位ms

返回值

None

在文件 **wdog.c** 第 **37** 行定义.

void WDOG_Refresh (void)

喂狗

```
1 WDOG_Refresh();   //喂狗
```

返回值

None

在文件 **wdog.c** 第 **173** 行定义.

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- a -

- ADC0_IRQHandler() : **adc.c**
- ADC1_IRQHandler() : **adc.c**
- ADC_Calibration() : **adc.c**
- ADC_CallbackInstall() : **adc.c**
- ADC_ChlMuxConfig() : **adc.c**
- ADC_EnableHardwareTrigger() : **adc.c**
- ADC_Init() : **adc.c**
- ADC_IsConversionCompleted() : **adc.c**
- ADC_ITDMAConfig() : **adc.c**
- ADC_QuickInit() : **adc.c**
- ADC_QuickReadValue() : **adc.c**
- ADC_ReadValue() : **adc.c**
- ADC_StartConversion() : **adc.c**
- assert_failed() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- b -

- BusFault_Handler() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- c -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- d -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- e -

- ENET_CallbackRxInstall() : **enet.c**
- ENET_CallbackTxInstall() : **enet.c**
- ENET_Init() : **enet.c**
- ENET_IsTxTransferComplete() : **enet.c**
- ENET_ITDMAConfig() : **enet.c**
- ENET_MacReceiveData() : **enet.c**
- ENET_MacSendData() : **enet.c**
- ENET_MII_Init() : **enet.c**
- ENET_MII_Read() : **enet.c**
- ENET_MII_Write() : **enet.c**
- ENET_Receive_IRQHandler() : **enet.c**
- ENET_Transmit_IRQHandler() : **enet.c**
- EnterSTOPMode() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- f -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- g -

- GetClock() : **common.c**
- GetUID() : **common.c**
- GPIO_CallbackInstall() : **gpio.c**
- GPIO_Init() : **gpio.c**
- GPIO_ITDMAConfig() : **gpio.c**
- GPIO_PinConfig() : **gpio.c**
- GPIO_QuickInit() : **gpio.c**
- GPIO_ReadBit() : **gpio.c**
- GPIO_ReadPort() : **gpio.c**
- GPIO_ResetBit() : **gpio.c**
- GPIO_SetBit() : **gpio.c**
- GPIO_ToggleBit() : **gpio.c**
- GPIO_WriteBit() : **gpio.c**
- GPIO_WritePort() : **gpio.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- h -

- HardFault_Handler() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- i -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- l -

- LPTimer_IRQHandler() : **lptmr.c**
- LPTMR_CallbackInstall() : **lptmr.c**
- LPTMR_ClearCounter() : **lptmr.c**
- LPTMR_ITDMAConfig() : **lptmr.c**
- LPTMR_PC_Init() : **lptmr.c**
- LPTMR_PC_QuickInit() : **lptmr.c**
- LPTMR_PC_ReadCounter() : **lptmr.c**
- LPTMR_TC_Init() : **lptmr.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- n -

- NFC_BlockErase() : **nfc.c**
- NFC_GetBufAddr() : **nfc.c**
- NFC_Init() : **nfc.c**
- NFC_PageProgram() : **nfc.c**
- NFC_PageRead() : **nfc.c**
- NFC_ReadFlashID() : **nfc.c**
- NFC_SendResetCmd() : **nfc.c**
- NMI_Handler() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- p -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- q -

- QuickInitDecode() : **common.c**
- QuickInitEncode() : **common.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- r -

- RTC_CallbackInstall() : **rtc.c**
- RTC_GetTAR() : **rtc.c**
- RTC_GetTime() : **rtc.c**
- RTC_GetTSR() : **rtc.c**
- RTC_GetWeek() : **rtc.c**
- RTC_Init() : **rtc.c**
- RTC_IRQHandler() : **rtc.c**
- RTC_IsTimeValid() : **rtc.c**
- RTC_ITDMAConfig() : **rtc.c**
- RTC_QuickInit() : **rtc.c**
- RTC_SetAlarm() : **rtc.c**
- RTC_SetCompensation() : **rtc.c**
- RTC_SetTime() : **rtc.c**
- RTC_SetTSR() : **rtc.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- s -

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- t -

- TSI0_IRQHandler() : **tsi.c**
- TSI_CallbackInstall() : **tsi.c**
- TSI_GetCounter() : **tsi.c**
- TSI_Init() : **tsi.c**
- TSI_ITDMAConfig() : **tsi.c**
- TSI_QuickInit() : **tsi.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- u -

- UART0_RX_TX_IRQHandler() : **uart.c**
- UART1_RX_TX_IRQHandler() : **uart.c**
- UART2_RX_TX_IRQHandler() : **uart.c**
- UART3_RX_TX_IRQHandler() : **uart.c**
- UART4_RX_TX_IRQHandler() : **uart.c**
- UART5_RX_TX_IRQHandler() : **uart.c**
- UART_CallbackRxInstall() : **uart.c**
- UART_CallbackTxInstall() : **uart.c**
- UART_DeInit() : **uart.c**
- UART_DMAGetRemainByte() : **uart.c**
- UART_DMASendByte() : **uart.c**
- UART_EnableRxFIFO() : **uart.c**
- UART_EnableTxFIFO() : **uart.c**
- UART_GetRxFIFOSize() : **uart.c**
- UART_GetTxFIFOSize() : **uart.c**
- UART_Init() : **uart.c**
- UART_ITDMAConfig() : **uart.c**
- UART_printf() : **uart.c**
- UART_QuickInit() : **uart.c**
- UART_ReadByte() : **uart.c**
- UART_SelectDebugInstance() : **uart.c**
- UART_SetDMATxMode() : **uart.c**
- UART_SetRxFIFOWatermark() : **uart.c**
- UART_SetTxFIFOWatermark() : **uart.c**
- UART_WriteByte() : **uart.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- v -

- VREF_DeInit() : **vref.c**
- VREF_Init() : **vref.c**
- VREF_QuickInit() : **vref.c**
- VREF_SetTrimValue() : **vref.c**

这里列出了所有文档化的函数,变量,宏,枚举和类型定义等,并附带其详细说明:

- w -

- Watchdog_IRQHandler() : **wdog.c**
- WDOG_CallbackInstall() : **wdog.c**
- WDOG_ClearResetCounter() : **wdog.c**
- WDOG_GetCurrentCounter() : **wdog.c**
- WDOG_GetResetCounter() : **wdog.c**
- WDOG_Init() : **wdog.c**
- WDOG_ITDMAConfig() : **wdog.c**
- WDOG_QuickInit() : **wdog.c**
- WDOG_Refresh() : **wdog.c**

- a -

- b -

- BusFault_Handler() : **common.c**

- c -

- d -

- e -

- ENET_CallbackRxInstall() : **enet.c**
- ENET_CallbackTxInstall() : **enet.c**
- ENET_Init() : **enet.c**
- ENET_IsTxTransferComplete() : **enet.c**
- ENET_ITDMAConfig() : **enet.c**
- ENET_MacReceiveData() : **enet.c**
- ENET_MacSendData() : **enet.c**
- ENET_MII_Init() : **enet.c**
- ENET_MII_Read() : **enet.c**
- ENET_MII_Write() : **enet.c**
- ENET_Receive_IRQHandler() : **enet.c**
- ENET_Transmit_IRQHandler() : **enet.c**
- EnterSTOPMode() : **common.c**

- f -

- g -

- GetClock() : **common.c**
- GetUID() : **common.c**
- GPIO_CallbackInstall() : **gpio.c**
- GPIO_Init() : **gpio.c**
- GPIO_ITDMAConfig() : **gpio.c**
- GPIO_PinConfig() : **gpio.c**
- GPIO_QuickInit() : **gpio.c**
- GPIO_ReadBit() : **gpio.c**
- GPIO_ReadPort() : **gpio.c**
- GPIO_ResetBit() : **gpio.c**
- GPIO_SetBit() : **gpio.c**
- GPIO_ToggleBit() : **gpio.c**
- GPIO_WriteBit() : **gpio.c**
- GPIO_WritePort() : **gpio.c**

- h -

- HardFault_Handler() : **common.c**

- i -

- l -

- n -

- p -

- q -

- QuickInitDecode() : **common.c**
- QuickInitEncode() : **common.c**

- r -

- s -

- t -

- TSI0_IRQHandler() : **tsi.c**
- TSI_CallbackInstall() : **tsi.c**
- TSI_GetCounter() : **tsi.c**
- TSI_Init() : **tsi.c**
- TSI_ITDMAConfig() : **tsi.c**
- TSI_QuickInit() : **tsi.c**

- u -

- v -

- VREF_DeInit() : **vref.c**
- VREF_Init() : **vref.c**
- VREF_QuickInit() : **vref.c**
- VREF_SetTrimValue() : **vref.c**

- w -