readme

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
common.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
flexcan.c	
flexio.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
lpit.c	Www.beyondcore.net http://upcmcu.taobao.com
lptmr.c	Www.beyondcore.net http://upcmcu.taobao.com
lpuart.c	Www.beyondcore.net http://upcmcu.taobao.com
mcg.c	
pit.c	Www.beyondcore.net http://upcmcu.taobao.com
rtc.c	Www.beyondcore.net http://upcmcu.taobao.com
scg.c	Www.beyondcore.net http://upcmcu.taobao.com
sdhc.c	Www.beyondcore.net http://upcmcu.taobao.com
sdramc.c	Www.beyondcore.net http://upcmcu.taobao.com
spi.c	Www.beyondcore.net http://upcmcu.taobao.com
tpm.c	Www.beyondcore.net http://upcmcu.taobao.com
uart.c	Www.beyondcore.net http://upcmcu.taobao.com

adc.c 文件参考

浏览源代码.

函数

void	**ADC_SetChlMux** (uint32_t instance, uint32_t mux)
uint32_t	**ADC_Init** (uint32_t MAP, ADC_Speed_t speed)
void	**ADC_SetTrigMode** (uint32_t instance, ADC_Trig_t trig)
int32_t	**ADC_SoftRead** (uint32_t instance, uint32_t chl)
void	**ADC_SoftTrigger** (uint32_t instance, uint32_t chl)
void	**ADC_SetIntMode** (uint32_t instance, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.03

在文件 **adc.c** 中定义.

函数说明

uint32_t ADC_Init (uint32_t MAP,
ADC_Speed_t speed
)

初始化配置AD模块

注解
None

参数
MAP AD模块引脚位图，详见adc.h文件
speed : AD转换速度

- kADC_SpeedHigh : 高速度
- kADC_SpeedMiddle : 中速度
- kADC_SpeedLow : 低速度

返回值
HW_ADC0 或HW_ADC1

在文件 **adc.c** 第 **68** 行定义.

void ADC_SetChlMux (uint32_t instance, uint32_t mux)

设置AD模块A或B通道模式

注解
None

参数
instance
- HW_ADC0 : ADC0模块
- HW_ADC1 : ADC1模块

mux
- kADC_ChlMuxA : AD模块A通道
- kADC_ChlMuxB : AD模块B通道

返回值
None

在文件 adc.c 第 51 行定义.

void ADC_SetIntMode (uint32_t instance, bool val)

设置AD中断模式

注解
None

参数
instance
- HW_ADC0 : ADC0模块
- HW_ADC1 : ADC1模块

val
- 0 : 关闭中断
- 1 : 开启中断

返回值
None

在文件 adc.c 第 186 行定义.

void ADC_SetTrigMode (uint32_t instance, ADC_Trig_t trig)

设置AD模块触发模式

注解
None

参数
instance
- HW_ADC0 : ADC0模块
- HW_ADC1 : ADC1模块

trig
- TrigSoft : AD模块软件触发
- TrigHard : AD模块硬件触发

返回值
None

在文件 **adc.c** 第 **134** 行定义.

int32_t ADC_SoftRead (uint32_t instance,
uint32_t chl
)

读取AD模块转换结果

注解
None

参数
instance
- HW_ADC0 : ADC0模块
- HW_ADC1 : ADC1模块

ch1 AD引脚通道

返回值
AD转换结果

在文件 **adc.c** 第 **150** 行定义.

void ADC_SoftTrigger (uint32_t instance,
uint32_t chl
)

软件触发AD转换

注解
None

参数
instance
- HW_ADC0 : ADC0模块
- HW_ADC1 : ADC1模块

ch1 AD引脚通道

返回值

None

在文件 **adc.c** 第 **169** 行定义.

common.c 文件参考

浏览源代码.

函数

uint32_t	**GetClock** (Clock_t clock)
void	**SetPinMux** (uint32_t instance, uint32_t pin, uint32_t mux)
void	**SetPinOpenDrain** (uint32_t instance, uint32_t pin, bool val)
void	**SetPinPull** (uint32_t instance, uint32_t pin, uint32_t val)
void	**DelayInit** (void)
void	**DelayMs** (uint32_t ms)
void	**SysTick_SetTime** (uint32_t us)
void	**DelayUs** (uint32_t us)
void	**SysTick_SetIntMode** (bool val)
void	**SystemSoftReset** (void)
uint32_t	**GetResetCause** (void)
void	**SetPowerMode** (uint32_t mode, bool enSleepOnExit)
uint32_t	**GetUID** (void)
void	**NMI_Handler** (void)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2015.6.21

在文件 **common.c** 中定义.

函数说明

void DelayInit (void)

延时初始化

注解

返回值
None

在文件 **common.c** 第 **243** 行定义.

void DelayMs (uint32_t ms)

MS级延时

参数

ms : 延时数 单位MS

返回值

None

在文件 **common.c** 第 **258** 行定义.

void DelayUs (uint32_t us)

US秒级延时

参数

ms : 延时数 单位US

返回值

None

在文件 **common.c** 第 **293** 行定义.

uint32_t GetClock (Clock_t clock)

读取系统时钟

参数

clock 系统时钟

- kCoreClock 内核时钟
- kBusClock 总线时钟(通常为内核时钟的一半)

返回值

系统时钟的频率 Hz

在文件 **common.c** 第 **108** 行定义.

uint32_t GetResetCause (void)

获得复位原因

注解

the reset cause number can refers to RM, RCM chapter

返回值

复位原因代码

在文件 **common.c** 第 **413** 行定义.

uint32_t GetUID (void)

获得芯片UID

参数

None

返回值

UID

在文件 **common.c** 第 **483** 行定义.

void NMI_Handler (void)

NMI handler.

返回值

None

在文件 **common.c** 第 **498** 行定义.

void SetPinMux (uint32_t instance, uint32_t pin, uint32_t mux)

设置引脚复用

参数

instance GPIO模块号

- HW_GPIOx GPIOx

pin 0-31

mux 复用选项 0-7

返回值

None

在文件 **common.c** 第 **162** 行定义.

void SetPinOpenDrain (uint32_t instance, uint32_t pin, bool val)

设置引脚是否开启 OpenDrain 模式

参数

instance GPIO模块号

- HW_GPIOA GPIOA

- HW_GPIOB GPIOB
- HW_GPIOC GPIOC
- HW_GPIOD GPIOD
- HW_GPIOE GPIOE

pin 引脚 0-31
val true or false

返回值
None

在文件 **common.c** 第 **181** 行定义.

void SetPinPull (uint32_t instance,
uint32_t pin,
uint32_t val
)

设置引脚上下拉配置

参数
instance GPIO模块号

- HW_GPIOA GPIOA
- HW_GPIOB GPIOB
- HW_GPIOC GPIOC
- HW_GPIOD GPIOD
- HW_GPIOE GPIOE

pin 引脚 0-31
val 上下拉配置

- 0 下拉
- 1 上拉
- others 浮空

返回值
None

在文件 **common.c** 第 **206** 行定义.

void SetPowerMode (uint32_t mode,
bool enSleepOnExit
)

设置系统功耗模式

参数
退出停止模式后是否自动再次进入停止模式

返回值
None

在文件 common.c 第 427 行定义.

void SystemSoftReset (void)

软件复位

返回值
None

在文件 common.c 第 403 行定义.

void SysTick_SetIntMode (bool val)

开启或关闭SysTick中断

参数
val true or false

返回值
None

在文件 common.c 第 309 行定义.

void SysTick_SetTime (uint32_t us)

设置SysTick定时器时间

参数
定时器时间，单位US

返回值
None

在文件 common.c 第 280 行定义.

dma.c 文件参考

浏览源代码.

函数

void	**DMA_Init** (DMA_Init_t *Init)
void	**DMA_Start** (uint8_t chl)
uint32_t	**DMA_IsTransDone** (uint8_t chl)
void	**DMA_SetTransCnt** (uint8_t chl, uint32_t val)
uint32_t	**DMA_GetTransCnt** (uint8_t chl)
void	**DMA_Stop** (uint8_t chl)
void	**DMA_SetIntMode** (uint8_t chl, DMA_Int_t mode, bool val)
uint32_t	**DMA_GetDestAddr** (uint8_t chl)
uint32_t	**DMA_GetSrcAddr** (uint8_t chl)
void	**DMA_SetDestAddr** (uint8_t chl, uint32_t addr)
void	**DMA_SetSrcAddr** (uint8_t chl, uint32_t addr)
void	**DMA_ClearIntFlag** (uint32_t chl)
void	**DMA_SetSrcMod** (uint8_t chl, DMA_Modulo_t sMod)
void	**DMA_SetDestMod** (uint8_t chl, DMA_Modulo_t dMod)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.04

在文件 **dma.c** 中定义.

函数说明

void DMA_ClearIntFlag (uint32_t chl)

清除DMA中断标志

注解
None

参数
chl : DMA通道

返回值

None

在文件 dma.c 第 249 行定义.

uint32_t DMA_GetDestAddr (uint8_t chl)

获得DMA目标传输地址

注解
None

参数
chl : DMA通道

返回值
返回目标地址

在文件 dma.c 第 203 行定义.

uint32_t DMA_GetSrcAddr (uint8_t chl)

获得DMA源地址

注解
None

参数
chl : DMA通道

返回值
返回源地址

在文件 dma.c 第 214 行定义.

uint32_t DMA_GetTransCnt (uint8_t chl)

获得DMA传输计数器

注解
None

参数
ch1 DMA通道

返回值
DMA传输字节个数

在文件 dma.c 第 155 行定义.

void DMA_Init (DMA_Init_t * Init)

初始化配置DMA模块

注解
None

参数
Init DMA配置参数结构体，详见dma.h文件

返回值
None

在文件 **dma.c** 第 **49** 行定义.

uint32_t DMA_IsTransDone (uint8_t chl)

检查DMA是否传输完成

注解
None

参数
ch1 DMA通道

返回值
CH_OK：传输完成；**CH_ERR**：传输错误

在文件 **dma.c** 第 **124** 行定义.

void DMA_SetDestAddr (uint8_t chl, uint32_t addr)

设置DMA目标地址

注解
None

参数
chl : DMA通道
addr : 目标地址

返回值
None

在文件 **dma.c** 第 **226** 行定义.

void DMA_SetDestMod (uint8_t chl, DMA_Modulo_t dMod)

设置DMA目标字节数

注解
None

参数

chl : DMA通道

sMod DMA目标字节数，详见dma.h文件

返回值

None

在文件 **dma.c** 第 **345** 行定义.

```
void DMA_SetIntMode ( uint8_t      chl,
                      DMA_Int_t    mode,
                      bool         val
                    )
```

设置DMA中断模式

注解

None

参数

chl : DMA通道

instance

- kDMAInt_All : DMA中断

val :

- 0 : 关闭中断
- 1 : 开启中断

返回值

None

在文件 **dma.c** 第 **182** 行定义.

```
void DMA_SetSrcAddr ( uint8_t   chl,
                      uint32_t  addr
                    )
```

设置DMA源地址

注解

None

参数

chl : DMA通道

addr : 源地址

返回值

None

在文件 **dma.c** 第 **238** 行定义.

void DMA_SetSrcMod (uint8_t chl, DMA_Modulo_t sMod)

设置DMA源字节数

注解
None

参数
chl : DMA通道
sMod DMA源字节数，详见dma.h文件

返回值
None

在文件 **dma.c** 第 **333** 行定义.

void DMA_SetTransCnt (uint8_t chl, uint32_t val)

设置DMA传输计数器

注解
None

参数
ch1 DMA通道
val DMA传输字节个数

返回值
None

在文件 **dma.c** 第 **143** 行定义.

void DMA_Start (uint8_t chl)

启动DMA的指定通过

开启DMA传输

注解
None

参数
Init DMA配置参数结构体，详见dma.h文件

返回值
None

注解
None

参数
ch1 DMA通道

返回值
None

在文件 **dma.c** 第 **112** 行定义.

void DMA_Stop (uint8_t chl)

暂停DMA传输

注解
None

参数
ch1 DMA通道

返回值
None

在文件 **dma.c** 第 **166** 行定义.

enet.c 文件参考

浏览源代码.

函数

void	**ENET_SetMacAddr** (uint32_t instance, uint8_t *mac)
void	**ENET_PHY_Init** (void)
uint32_t	**ENET_PHY_Write** (uint16_t phy_addr, uint16_t reg_addr, uint16_t data)
uint32_t	**ENET_PHY_Read** (uint16_t phy_addr, uint16_t reg_addr, uint16_t *data)
void	**ENET_Init** (uint32_t MAP, uint8_t *mac)
uint32_t	**ENET_SendData** (uint8_t *data, uint16_t len)
uint32_t	**ENET_ReceiveData** (uint8_t *data)

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_Init (uint32_t MAP,
uint8_t * mac
)

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 **272** 行定义.

void ENET_PHY_Init (void)

初始化以太网 MII配置层接口

返回值
None

在文件 enet.c 第 145 行定义.

uint32_t ENET_PHY_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 226 行定义.

uint32_t ENET_PHY_Write (uint16_t phy_addr,
uint16_t reg_addr,
uint16_t data
)

enet physical controller write data

参数
[in] phy_addr phy addr
[in] reg_addr regisrer adderss
[in] data data

返回值
CH_OK : succ
others failed

在文件 enet.c 第 177 行定义.

uint32_t ENET_ReceiveData (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 **381** 行定义.

---

uint32_t ENET_SendData (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 **330** 行定义.

---

void ENET_SetMacAddr (uint32_t instance,
uint8_t * mac
)

设置ENET模块的接收MAC地址

注解

内部函数

参数

[in] **pa** MAC地址

返回值

None

在文件 **enet.c** 第 **118** 行定义.

flash.c 文件参考

浏览源代码.

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_Test** (uint32_t addr, uint32_t len)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.6.6

在文件 **flash.c** 中定义.

函数说明

uint8_t FLASH_EraseSector (uint32_t addr)

擦除Flash扇区

注解
该功能将删除一个Flash扇区的内容

参数
addr 擦除区域起始地址

返回值
返回操作结果

在文件 **flash.c** 第 **109** 行定义.

uint32_t FLASH_GetSectorSize (void)

获得扇区大小

返回值
Flash扇区尺寸

在文件 **flash.c** 第 **88** 行定义.

void FLASH_Init (void)

初始化Flash

返回值

Flash扇区尺寸

在文件 **flash.c** 第 **97** 行定义.

uint32_t FLASH_Test (uint32_t addr, uint32_t len)

Flash自测

注解

确保有足够的栈空间

参数

addr 开始地址

返回值

FLASH_OK:成功；其它：测试错误

在文件 **flash.c** 第 **289** 行定义.

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

写Flash一个扇区

注解

字节数必须等于扇区尺寸

参数

addr 开始地址

buf ：写入数据起始指针

len ：字节数

返回值

返回执行结果

在文件 **flash.c** 第 **139** 行定义.

flexbus.c 文件参考

浏览源代码.

函数

void	**FLEXBUS_Init** (FLEXBUS_Init_t *Init)
void	**FLEXBUS_TimingConfig** (uint32_t CS, FLEXBUS_TimingConfig_t *config)

详细描述

作者
YANDLD

版本
V3.0

日期
2016.2.20

在文件 **flexbus.c** 中定义.

函数说明

void FLEXBUS_Init (FLEXBUS_Init_t * Init)

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 **21** 行定义.

void FLEXBUS_TimingConfig (uint32_t CS, FLEXBUS_TimingConfig_t * config)

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 **78** 行定义.

flexcan.c

```
  1
 12 #include "flexcan.h"
 13 #include "gpio.h"
 14
 15 #if defined(CAN0)
 16
 17 #define CAN_MB_MAX      (ARRAY_SIZE(CAN0->MB))
 18
 19 #if (!defined(CAN_BASES))
 20 #if defined(CAN1)
 21 #define CAN_BASES   {CAN0, CAN1}
 22 #else
 23 #define CAN_BASES   {CAN0}
 24 #endif
 25 #endif
 26
 27 /* global vars */
 28 CAN_Type * const FLEXCANBases[] = CAN_BASES;
 29
 30 static const Reg_t ClkTbl[] =
 31 {
 32     {(void*)&(SIM->SCGC6), SIM_SCGC6_FLEXCAN0_MASK},
 33 #ifdef CAN1
 34     #if defined(SIM_SCGC3_FLEXCAN1_MASK)
 35     {(void*)&(SIM->SCGC3), SIM_SCGC3_FLEXCAN1_MASK},
 36     #elif defined(SIM_SCGC6_FLEXCAN1_MASK)
 37     {(void*)&(SIM->SCGC6), SIM_SCGC6_FLEXCAN1_MASK},
 38     #endif
 39 #endif
 40 };
 41
 42 static const IRQn_Type FLEXCAN_IRQTbl[] =
 43 {
 44     CAN0_ORed_Message_buffer_IRQn,
 45 #ifdef CAN1
 46     CAN1_ORed_Message_buffer_IRQn,
 47 #endif
 48 };
 49
 50 /* CAN MB Type */
 51 typedef enum
 52 {
 53     kFlexCanTX_Inactive  = 0x08,
 54     kFlexCanTX_Abort     = 0x09,
 55     kFlexCanTX_Data      = 0x0C,
 56     kFlexCanTX_Remote    = 0x1C,
 57     kFlexCanTX_Tanswer   = 0x0E,
 59     kFlexCanTX_NotUsed   = 0xF,
 60     kFlexCanRX_Inactive  = 0x0,
 61     kFlexCanRX_Full      = 0x2,
 62     kFlexCanRX_Empty     = 0x4,
 63     kFlexCanRX_Overrun   = 0x6,
 64     kFlexCanRX_Busy      = 0x8,
 66     kFlexCanRX_Ranswer   = 0xA,
 68     kFlexCanRX_NotUsed   = 0xF,
 69 }CAN_MBCode_Type;
 70
 71 #define CAN_GET_MB_CODE(cs)         (((cs) & CAN_CS_CODE_MASK)>>
    CAN_CS_CODE_SHIFT)
 72
 73
 83 static uint32_t CAN_SetBaudrate(uint32_t instance, uint32_t
    baudrate)
 84 {
 85     uint32_t ps ,p1, p2, pd;
 86     uint32_t sclock = GetClock(kBusClock)/baudrate;
```

```
 87      uint32_t time_seg1, time_seg2, total_tq;
 88      CAN_Type *CANx = FLEXCANBases[instance];
 89
 90      for(pd=0xFF; pd>0; pd--)
 91      {
 92          for(ps=1; ps<8; ps++)
 93          {
 94              for(p1=1; p1<8; p1++)
 95              {
 96                  for(p2=1; p2<8; p2++)
 97                  {
 98                      time_seg1 = ps+p1+2;
 99                      time_seg2 = p2+1;
100                      total_tq = time_seg1+time_seg2+1;
101
102                      if(ABS((int32_t)(total_tq*(pd+1) - sclock)) <
    2)
103                      {
104                          if((time_seg1 < (time_seg2+8)) &&
    ((time_seg2+2) > time_seg1))
105                          {
106                              CANx->CTRL1 &=
    ~(CAN_CTRL1_PROPSEG_MASK | CAN_CTRL1_RJW_MASK |
    CAN_CTRL1_PSEG1_MASK | CAN_CTRL1_PSEG2_MASK |
    CAN_CTRL1_PRESDIV_MASK);
107                              CANx->CTRL1 |=  CAN_CTRL1_PROPSEG(ps)
108                                           | CAN_CTRL1_RJW(2)
109                                           | CAN_CTRL1_PSEG1(p1)
110                                           | CAN_CTRL1_PSEG2(p2)
111                                           |
    CAN_CTRL1_PRESDIV(pd);
112                              LIB_TRACE("Get baudrate param! pd %d
    ps %d p1 %d p2 %d\r\n", pd, ps,p1,p2);
113                              return CH_OK;
114                          }
115                      }
116                  }
117              }
118          }
119      }
120      return CH_ERR;
121  }
122
133  void CAN_SetRxFilterMask(uint32_t instance, uint32_t mb, uint32_t
    mask)
134  {
135      FLEXCANBases[instance]->MCR |= (CAN_MCR_FRZ_MASK |
    CAN_MCR_HALT_MASK);
136      while(!(CAN_MCR_FRZACK_MASK & (FLEXCANBases[instance]->MCR)))
    {};
137
138      if(mask > 0x7FF)
139      {
140          FLEXCANBases[instance]->RXIMR[mb] = CAN_ID_EXT(mask);
141      }
142      else
143      {
144          FLEXCANBases[instance]->RXIMR[mb] = CAN_ID_STD(mask);
145      }
146
147      FLEXCANBases[instance]->MCR &= ~(CAN_MCR_FRZ_MASK |
    CAN_MCR_HALT_MASK);
148      while((CAN_MCR_FRZACK_MASK & (FLEXCANBases[instance]->MCR)));
149  }
150
161  void CAN_SetRxMB(uint32_t instance, uint32_t mb, uint32_t id)
162  {
163      FLEXCANBases[instance]->MB[mb].WORD0 = 0;
164      FLEXCANBases[instance]->MB[mb].WORD1 = 0;
165      FLEXCANBases[instance]->MB[mb].CS = 0;
```

```
166     if(id > 0x7FF)
167     {
168         FLEXCANBases[instance]->MB[mb].ID = id;  /* ID [28-0]*/
169         FLEXCANBases[instance]->MB[mb].CS |= (CAN_CS_SRR_MASK |
    CAN_CS_IDE_MASK);
170     }
171     else
172     {
173         FLEXCANBases[instance]->MB[mb].ID = CAN_ID_STD(id);  /*
    ID[28-18] */
174         FLEXCANBases[instance]->MB[mb].CS &= ~(CAN_CS_IDE_MASK |
    CAN_CS_SRR_MASK);
175     }
176
177     FLEXCANBases[instance]->MB[mb].CS |=
    CAN_CS_CODE(kFlexCanRX_Empty) | CAN_CS_RTR_MASK;
178 }
179
189 void CAN_SetTxMB(uint32_t instance, uint32_t mb)
190 {
191     FLEXCANBases[instance]->MB[mb].CS =
    CAN_CS_CODE(kFlexCanTX_Inactive);
192     FLEXCANBases[instance]->MB[mb].ID = 0x0;
193     FLEXCANBases[instance]->MB[mb].WORD0 = 0x0;
194     FLEXCANBases[instance]->MB[mb].WORD1 = 0x0;
195 }
196
205 uint32_t CAN_Init(uint32_t MAP, uint32_t baudrate)
206 {
207     uint32_t i;
208     CAN_Type *CANx;
209     map_t * pq = (map_t*)&(MAP);
210
211     REG_SET(ClkTbl, pq->ip);
212     CANx = FLEXCANBases[pq->ip];
213
214     /* set clock source is bus clock */
215     CANx->CTRL1 |= CAN_CTRL1_CLKSRC_MASK;
216
217     /* enable module */
218     CANx->MCR &= ~CAN_MCR_MDIS_MASK;
219
220     /* software reset */
221     CANx->MCR |= CAN_MCR_SOFTRST_MASK;
222     while(CAN_MCR_SOFTRST_MASK & (CANx->MCR)) {};
223
224     /* halt mode */
225     CANx->MCR |= (CAN_MCR_FRZ_MASK | CAN_MCR_HALT_MASK);
226     while(!(CAN_MCR_FRZACK_MASK & (CANx->MCR))) {};
227
228     /* init all mb */
229     for(i=0; i<CAN_MB_MAX; i++)
230     {
231         CANx->MB[i].CS = 0x00000000;
232         CANx->MB[i].ID = 0x00000000;
233         CANx->MB[i].WORD0 = 0x00000000;
234         CANx->MB[i].WORD1 = 0x00000000;
235         CANx->RXIMR[i] = 0x00000000; /* received all frame */
236         CANx->IMASK1 = 0x00000000;
237         CANx->IFLAG1 = 0xFFFFFFFF;
238     }
239
240     CANx->CTRL2 = CAN_CTRL2_TASD(0x16) | CAN_CTRL2_RRS_MASK |
    CAN_CTRL2_EACEN_MASK;
241
242     /* set all masks */
243     //CANx->RXMGMASK = CAN_ID_EXT(CAN_RXMGMASK_MG_MASK);
244    // CANx->RX14MASK = CAN_ID_EXT(CAN_RX14MASK_RX14M_MASK);
245    // CANx->RX15MASK = CAN_ID_EXT(CAN_RX15MASK_RX15M_MASK);
246     /* use indviual mask, do not use RXMGMASK, RX14MASK and
```

```
    RX15MASK */
247     CANx->MCR |= CAN_MCR_IRMQ_MASK;
248     CANx->MCR &= ~CAN_MCR_IDAM_MASK;
249
250     /* setting baudrate */
251     CAN_SetBaudrate(pq->ip, baudrate);
252
253     /* bypass the frame sended by itself */
254     CANx->MCR |= CAN_MCR_SRXDIS_MASK;
255
256     /* enable module */
257     CANx->MCR &= ~(CAN_MCR_FRZ_MASK | CAN_MCR_HALT_MASK);
258     while((CAN_MCR_FRZACK_MASK & (CANx->MCR)));
259     while((CANx->MCR)&CAN_MCR_NOTRDY_MASK);
260
261     PIN_SET_MUX;
262     return CH_OK;
263 }
264
277 uint32_t CAN_SendDataFrame(uint32_t instance, uint32_t mb,
    uint32_t id, uint8_t* buf, uint8_t len)
278 {
279     uint32_t i, cs_temp, timeout, word[2];
280
281     word[0] = 0;
282     word[1] = 0;
283     cs_temp = 0;
284     timeout = 0;
285
286     while((FLEXCANBases[instance]->MB[mb].CS & CAN_CS_CODE_MASK)
    != CAN_CS_CODE(kFlexCanTX_Inactive) && timeout < 100000)
287     {
288         timeout++;
289     }
290     if(timeout == 100000)
291     {
292         LIB_TRACE("CAN_SendDataFrame timeout!\r\n");
293         return CH_TIMEOUT;
294     }
295
296     /* setting data */
297     for(i=0; i<len; i++)
298     {
299         (i<4)?( word[0] |= (*(buf+i)<<((3-i)*8))):(word[1] |=
    (*(buf+i)<<((7-i)*8)));
300     }
301
302     FLEXCANBases[instance]->MB[mb].WORD0 = word[0];
303     FLEXCANBases[instance]->MB[mb].WORD1 = word[1];
304
305     /* len field */
306     cs_temp |= CAN_CS_DLC(len);
307
308     /* ID and IDE */
309     if(id > 0x7FF)
310     {
311         FLEXCANBases[instance]->MB[mb].ID = id;  /* ID [28-0]*/
312         cs_temp |= (CAN_CS_SRR_MASK | CAN_CS_IDE_MASK);
313     }
314     else
315     {
316         FLEXCANBases[instance]->MB[mb].ID = CAN_ID_STD(id);  /*
    ID[28-18] */
317     }
318
319     cs_temp |= CAN_CS_CODE(kFlexCanTX_Data);
320
321     FLEXCANBases[instance]->MB[mb].CS = cs_temp;
322     return CH_OK;
323 }
```

```
324
338 uint32_t CAN_SendRemoteFrame(uint32_t instance, uint32_t mb,
    uint32_t id, uint8_t req_len)
339 {
340     uint32_t cs_temp, timeout;
341
342     cs_temp = 0;
343     timeout = 0;
344
345     while((FLEXCANBases[instance]->MB[mb].CS & CAN_CS_CODE_MASK)
    != CAN_CS_CODE(kFlexCanTX_Inactive) && timeout < 100000)
346     {
347         timeout++;
348     }
349
350     if(timeout == 100000)
351     {
352         LIB_TRACE("CAN_SendRemoteFrame timeout!\r\n");
353         return CH_TIMEOUT;
354     }
355
356     /* DLC field, remote frame still has DLC filed, it's request
    len */
357     cs_temp |= CAN_CS_DLC(req_len) | CAN_CS_RTR_MASK;
358
359     /* ID and IDE */
360     if(id > 0x7FF)
361     {
362         FLEXCANBases[instance]->MB[mb].ID = id;  /* ID [28-0]*/
363         cs_temp |= (CAN_CS_SRR_MASK | CAN_CS_IDE_MASK);
364     }
365     else
366     {
367         FLEXCANBases[instance]->MB[mb].ID = CAN_ID_STD(id);  /*
    ID[28-18] */
368     }
369
370     cs_temp |= CAN_CS_CODE(kFlexCanTX_Remote);
371     FLEXCANBases[instance]->MB[mb].CS = cs_temp;
372
373     return CH_OK;
374 }
375
391 void CAN_SetIntMode(uint32_t instance, uint32_t mb, CAN_Int_t
    mode, bool val)
392 {
393     NVIC_EnableIRQ(FLEXCAN_IRQTbl[instance]);
394     switch(mode)
395     {
396         case kCAN_IntTx:
397         case kCAN_IntRx:
398             (val)?(FLEXCANBases[instance]->IMASK1 |= (1 <<
    mb)):(FLEXCANBases[instance]->IMASK1 &= ~(1 << mb));
399             break;
400         default:
401             break;
402     }
403 }
404
418 uint32_t CAN_ReceiveFrame(uint32_t instance, uint32_t mb,
    uint32_t *id, uint8_t *buf, uint8_t *len, bool *isRemote)
419 {
420     uint32_t code, i;
421     uint32_t word[2] = {0};
422     uint32_t cs_temp;
423     cs_temp = FLEXCANBases[instance]->MB[mb].CS;
424     code = CAN_GET_MB_CODE(cs_temp);
425
426     if(code & 0x01)
427     {
```

```
428             return CH_IO_ERR; /* MB is busy and controlled by hardware */
429         }
430
431         if(FLEXCANBases[instance]->IFLAG1 & (1<<mb))
432         {
433             /* clear IT pending bit */
434             FLEXCANBases[instance]->IFLAG1 = (1 << mb);
435
436             /* read content */
437             *len = (cs_temp & CAN_CS_DLC_MASK) >> CAN_CS_DLC_SHIFT;
438             word[0] = FLEXCANBases[instance]->MB[mb].WORD0;
439             word[1] = FLEXCANBases[instance]->MB[mb].WORD1;
440
441             (cs_temp & CAN_CS_RTR_MASK)?(*isRemote = true):(*isRemote = false);
442
443             for(i = 0; i < *len; i++)
444             {
445                 (i < 4)?(*(buf + i))=(word[0]>>((3-i)*8)):((*(buf + i))=(word[1]>>((7-i)*8)));
446             }
447
448             *id = (FLEXCANBases[instance]->MB[mb].ID & (CAN_ID_EXT_MASK | CAN_ID_STD_MASK));
449
450             if(!(cs_temp & CAN_CS_IDE_MASK))
451             {
452                 *id >>= CAN_ID_STD_SHIFT;
453             }
454
455             i = FLEXCANBases[instance]->TIMER; /* unlock MB */
456             return CH_OK;
457         }
458         i = FLEXCANBases[instance]->TIMER; /* unlock MB */
459         return CH_IO_ERR;
460 }
461
472 uint32_t CAN_ReadFIFO(uint32_t instance, uint32_t *id, uint8_t *buf, uint8_t *len)
473 {
474     uint32_t i;
475     uint32_t word[2] = {0};
476
477     /* read content */
478     *len = (FLEXCANBases[instance]->MB[0].CS & CAN_CS_DLC_MASK) >> CAN_CS_DLC_SHIFT;
479     word[0] = FLEXCANBases[instance]->MB[0].WORD0;
480     word[1] = FLEXCANBases[instance]->MB[0].WORD1;
481     for(i = 0; i < *len; i++)
482     {
483         if(i < 4)
484         (*(buf + i))=(word[0]>>((3-i)*8));
485         else
486         (*(buf + i))=(word[1]>>((7-i)*8));
487     }
488     *id = (FLEXCANBases[instance]->MB[0].ID & (CAN_ID_EXT_MASK | CAN_ID_STD_MASK));
489     i = FLEXCANBases[instance]->TIMER; /* unlock MB */
490     FLEXCANBases[instance]->IFLAG1 = (1 << CAN_RX_FIFO_MB);
491     return 0;
492 }
493
502 bool CAN_IsRxFIFOEnable(uint32_t instance)
503 {
504     return (FLEXCANBases[instance]->MCR & CAN_MCR_RFEN_MASK);
505 }
506
517 void CAN_SetRxFIFO(uint32_t instance, bool val)
518 {
```

```
519     CAN_Type *CANx;
520
521     CANx = FLEXCANBases[instance];
522     /* halt mode */
523     CANx->MCR |= (CAN_MCR_FRZ_MASK | CAN_MCR_HALT_MASK);
524     while(!(CAN_MCR_FRZACK_MASK & (CANx->MCR))) {};
525
526     (val == true)?(CANx->MCR |= CAN_MCR_RFEN_MASK):(CANx->MCR &=
    ~CAN_MCR_RFEN_MASK);
527     CANx->CTRL2 &= ~CAN_CTRL2_RFFN_MASK;
528     CANx->CTRL2 |= CAN_CTRL2_RFFN(0);
529
530     /* enable module */
531     CANx->MCR &= ~(CAN_MCR_FRZ_MASK | CAN_MCR_HALT_MASK);
532     while((CAN_MCR_FRZACK_MASK & (CANx->MCR)));
533     while(((CANx->MCR)&CAN_MCR_NOTRDY_MASK));
534 }
535
536
537
538 #endif
```

flexio.c 文件参考

浏览源代码.

函数

void	**FLEXIO_SetShifterConfig** (FLEXIO_Type *base, uint8_t index, const **flexio_shifter_config_t** *shifterConfig)
void	**FLEXIO_SetTimerConfig** (FLEXIO_Type *base, uint8_t index, const **flexio_timer_config_t** *timerConfig)
void	**FLEXIO_Init** (void)
void	**FLEXIO_PWM_Init** (uint32_t chl, uint32_t pin, uint32_t freq)
void	**FLEXIO_PWM_Start** (uint32_t chl, uint32_t pin)
void	**FLEXIO_PWM_Stop** (uint32_t chl, uint32_t pin)
void	**FLEXIO_UART_Init** (uint32_t instance, uint32_t baudrate, uint32_t tx_pin, uint32_t rx_pin)
void	**FLEXIO_UART_PutChar** (uint32_t instance, uint8_t ch)
uint8_t	**FLEXIO_UART_GetChar** (uint32_t instance)

详细描述

作者
YANDLD

版本
V3.0

日期
2016.6.13

在文件 **flexio.c** 中定义.

枚举类型说明

enum _flexio_pin_config

Define type of timer/shifter pin configuration.

枚举值	
kFLEXIO_PinConfigOutputDisabled	Pin output disabled.
kFLEXIO_PinConfigOpenDrainOrBidirection	Pin open drain or bidirectional output enable.
kFLEXIO_PinConfigBidirectionOutputData	Pin bidirectional output data.
kFLEXIO_PinConfigOutput	Pin output.

在文件 **flexio.c** 第 **38** 行定义.

enum _flexio_pin_polarity

Definition of pin polarity.

枚举值	
kFLEXIO_PinActiveHigh	Active high.
kFLEXIO_PinActiveLow	Active low.

在文件 **flexio.c** 第 **47** 行定义.

enum _flexio_shifter_buffer_type

Define FlexIO shifter buffer type.

枚举值	
kFLEXIO_ShifterBuffer	Shifter Buffer N Register.
kFLEXIO_ShifterBufferBitSwapped	Shifter Buffer N Bit Byte Swapped Register.
kFLEXIO_ShifterBufferByteSwapped	Shifter Buffer N Byte Swapped Register.
kFLEXIO_ShifterBufferBitByteSwapped	Shifter Buffer N Bit Swapped Register.
kFLEXIO_ShifterBufferNibbleByteSwapped	Shifter Buffer N Nibble Byte Swapped Register.
kFLEXIO_ShifterBufferHalfWordSwapped	Shifter Buffer N Half Word Swapped Register.
kFLEXIO_ShifterBufferNibbleSwapped	Shifter Buffer N Nibble Swapped Register.

在文件 **flexio.c** 第 **193** 行定义.

enum _flexio_shifter_input_source

Define type of shifter input source.

枚举值	
kFLEXIO_ShifterInputFromPin	Shifter input from pin.
kFLEXIO_ShifterInputFromNextShifterOutput	Shifter input from Shifter N+1.

在文件 **flexio.c** 第 **167** 行定义.

enum _flexio_shifter_mode

Define type of shifter working mode.

枚举值	
kFLEXIO_ShifterDisabled	Shifter is disabled.
kFLEXIO_ShifterModeReceive	Receive mode.
kFLEXIO_ShifterModeTransmit	Transmit mode.
kFLEXIO_ShifterModeMatchStore	Match store mode.
kFLEXIO_ShifterModeMatchContinuous	Match continuous mode.
kFLEXIO_ShifterModeState	SHIFTBUF contents are used for storing programmable state attributes.

kFLEXIO_ShifterModeLogic	SHIFTBUF contents are used for implementing programmable logic look up table.

在文件 **flexio.c** 第 **149** 行定义.

enum _flexio_shifter_start_bit

Define type of START bit configuration.

枚举值	
kFLEXIO_ShifterStartBitDisabledLoadDataOnEnable	Disable shifter start bit, transmitter loads data on enable.
kFLEXIO_ShifterStartBitDisabledLoadDataOnShift	Disable shifter start bit, transmitter loads data on first shift.
kFLEXIO_ShifterStartBitLow	Set shifter start bit to logic low level.
kFLEXIO_ShifterStartBitHigh	Set shifter start bit to logic high level.

在文件 **flexio.c** 第 **182** 行定义.

enum _flexio_shifter_stop_bit

Define of STOP bit configuration.

枚举值	
kFLEXIO_ShifterStopBitDisable	Disable shifter stop bit.
kFLEXIO_ShifterStopBitLow	Set shifter stop bit to logic low level.
kFLEXIO_ShifterStopBitHigh	Set shifter stop bit to logic high level.

在文件 **flexio.c** 第 **174** 行定义.

enum _flexio_timer_decrement_source

Define type of timer decrement.

枚举值	
kFLEXIO_TimerDecSrcOnFlexIOClockShiftTimerOutput	Decrement counter on FlexIO clock, Shift clock equals Timer output.
kFLEXIO_TimerDecSrcOnTriggerInputShiftTimerOutput	Decrement counter on Trigger input (both edges), Shift clock equals Timer output.
kFLEXIO_TimerDecSrcOnPinInputShiftPinInput	Decrement counter on Pin input (both edges), Shift clock equals Pin input.
kFLEXIO_TimerDecSrcOnTriggerInputShiftTriggerInput	Decrement counter on Trigger input (both edges), Shift clock equals Trigger input.

在文件 **flexio.c** 第 **74** 行定义.

enum _flexio_timer_disable_condition

Define type of timer disable condition.

枚举值	
kFLEXIO_TimerDisableNever	Timer never disabled.
kFLEXIO_TimerDisableOnPreTimerDisable	Timer disabled on Timer N-1 disable.
kFLEXIO_TimerDisableOnTimerCompare	Timer disabled on Timer compare.
kFLEXIO_TimerDisableOnTimerCompareTriggerLow	Timer disabled on Timer compare and Trigger Low.
kFLEXIO_TimerDisableOnPinBothEdge	Timer disabled on Pin rising or falling edge.
kFLEXIO_TimerDisableOnPinBothEdgeTriggerHigh	Timer disabled on Pin rising or falling edge provided Trigger is high.
kFLEXIO_TimerDisableOnTriggerFallingEdge	Timer disabled on Trigger falling edge.

在文件 **flexio.c** 第 **99** 行定义.

enum _flexio_timer_enable_condition

Define type of timer enable condition.

枚举值	
kFLEXIO_TimerEnabledAlways	Timer always enabled.
kFLEXIO_TimerEnableOnPrevTimerEnable	Timer enabled on Timer N-1 enable.
kFLEXIO_TimerEnableOnTriggerHigh	Timer enabled on Trigger high.
kFLEXIO_TimerEnableOnTriggerHighPinHigh	Timer enabled on Trigger high and Pin high.
kFLEXIO_TimerEnableOnPinRisingEdge	Timer enabled on Pin rising edge.
kFLEXIO_TimerEnableOnPinRisingEdgeTriggerHigh	Timer enabled on Pin rising edge and Trigger high.
kFLEXIO_TimerEnableOnTriggerRisingEdge	Timer enabled on Trigger rising edge.
kFLEXIO_TimerEnableOnTriggerBothEdge	Timer enabled on Trigger rising or falling edge.

在文件 **flexio.c** 第 **112** 行定义.

enum _flexio_timer_mode

Define type of timer work mode.

枚举值	
kFLEXIO_TimerModeDisabled	Timer Disabled.
kFLEXIO_TimerModeDual8BitBaudBit	Dual 8-bit counters baud/bit mode.
kFLEXIO_TimerModeDual8BitPWM	Dual 8-bit counters PWM mode.
kFLEXIO_TimerModeSingle16Bit	Single 16-bit counter mode.

在文件 **flexio.c** 第 **54** 行定义.

enum _flexio_timer_output

Define type of timer initial output or timer reset condition.

枚举值	
kFLEXIO_TimerOutputOneNotAffectedByReset	Logic one when enabled and is not affected by timer reset.
kFLEXIO_TimerOutputZeroNotAffectedByReset	Logic zero when enabled and is not affected by timer reset.
kFLEXIO_TimerOutputOneAffectedByReset	Logic one when enabled and on timer reset.
kFLEXIO_TimerOutputZeroAffectedByReset	Logic zero when enabled and on timer reset.

在文件 **flexio.c** 第 **63** 行定义.

enum _flexio_timer_reset_condition

Define type of timer reset condition.

枚举值	
kFLEXIO_TimerResetNever	Timer never reset.
kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput	Timer reset on Timer Pin equal to Timer Output.
kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput	Timer reset on Timer Trigger equal to Timer Output.
kFLEXIO_TimerResetOnTimerPinRisingEdge	Timer reset on Timer Pin rising edge.
kFLEXIO_TimerResetOnTimerTriggerRisingEdge	Timer reset on Trigger rising edge.
kFLEXIO_TimerResetOnTimerTriggerBothEdge	Timer reset on Trigger rising or falling edge.

在文件 **flexio.c** 第 **87** 行定义.

enum _flexio_timer_start_bit_condition

Define type of timer start bit generate condition.

枚举值	
kFLEXIO_TimerStartBitDisabled	Start bit disabled.
kFLEXIO_TimerStartBitEnabled	Start bit enabled.

在文件 **flexio.c** 第 **135** 行定义.

enum _flexio_timer_stop_bit_condition

Define type of timer stop bit generate condition.

枚举值	
kFLEXIO_TimerStopBitDisabled	Stop bit disabled.
kFLEXIO_TimerStopBitEnableOnTimerCompare	Stop bit is enabled on timer compare.
kFLEXIO_TimerStopBitEnableOnTimerDisable	Stop bit is enabled on timer disable.
kFLEXIO_TimerStopBitEnableOnTimerCompareDisable	Stop bit is enabled on timer compare and timer disable.

在文件 **flexio.c** 第 **125** 行定义.

enum _flexio_timer_trigger_polarity

Define time of timer trigger polarity.

枚举值	
kFLEXIO_TimerTriggerPolarityActiveHigh	Active high.
kFLEXIO_TimerTriggerPolarityActiveLow	Active low.

在文件 **flexio.c** 第 **24** 行定义.

enum _flexio_timer_trigger_source

Define type of timer trigger source.

枚举值	
kFLEXIO_TimerTriggerSourceExternal	External trigger selected.
kFLEXIO_TimerTriggerSourceInternal	Internal trigger selected.

在文件 **flexio.c** 第 **31** 行定义.

函数说明

void FLEXIO_Init (void)

初始化配置FlexIO模块

注解
None

在文件 **flexio.c** 第 **319** 行定义.

void FLEXIO_PWM_Init (uint32_t chl, uint32_t pin, uint32_t freq)

初始化配置FlexIO模块在PWM模式

注解
None

参数
chl ：pwm通道0或1
pin ：引脚号
freg：pwm波形频率

返回值
None

在文件 flexio.c 第 340 行定义.

```
void FLEXIO_PWM_Start ( uint32_t chl,
                        uint32_t pin
                      )
```

开启PWM波形输出

注解
None

参数
chl ：pwm通道0或1
pin ：引脚号

返回值
None

在文件 flexio.c 第 368 行定义.

```
void FLEXIO_PWM_Stop ( uint32_t chl,
                       uint32_t pin
                     )
```

关闭PWM波形输出

注解
None

参数
chl ：pwm通道0或1
pin ：引脚号

返回值
None

在文件 flexio.c 第 381 行定义.

```
void FLEXIO_SetShifterConfig ( FLEXIO_Type *                 base,
                               uint8_t                       index,
                               const flexio_shifter_config_t * shifterConfig
```

)

设置移位

注解
None

在文件 **flexio.c** 第 **279** 行定义.

void FLEXIO_SetTimerConfig (FLEXIO_Type * base,
uint8_t index,
const flexio_timer_config_t * timerConfig
)

设置定时器

注解
None

在文件 **flexio.c** 第 **298** 行定义.

uint8_t FLEXIO_UART_GetChar (uint32_t instance)

串口接收一个字符

注解
None

参数
instance ： HW_FLEXIO_UART0或HW_FLEXIO_UART1

返回值
返回接收到的字符

在文件 **flexio.c** 第 **505** 行定义.

void FLEXIO_UART_Init (uint32_t instance,
uint32_t baudrate,
uint32_t tx_pin,
uint32_t rx_pin
)

初始化配置FlexIO模块在串口模式

注解
None

参数
instance ： HW_FLEXIO_UART0或HW_FLEXIO_UART1
baudrate ： 通信波特率
tx_pin ：发送引脚
rx_pin ：接收引脚

返回值
None

在文件 flexio.c 第 396 行定义.

void FLEXIO_UART_PutChar (uint32_t instance, uint8_t ch)

串口发送一个字符

注解
None

参数
instance ： HW_FLEXIO_UART0或HW_FLEXIO_UART1
ch 需要发送的字符

返回值
None

在文件 flexio.c 第 489 行定义.

ftm.c 文件参考

浏览源代码.

函数

void	**FTM_PWM_SetDuty** (uint32_t instance, uint8_t chl, uint32_t duty)
uint32_t	**FTM_GetChlCounter** (uint32_t instance, uint32_t chl)
void	**FTM_SetMoudlo** (uint32_t instance, uint32_t val)
uint32_t	**FTM_GetMoudlo** (uint32_t instance)

详细描述

作者
YANDLD

版本
V3.0

日期
2016.5.09

在文件 **ftm.c** 中定义.

函数说明

uint32_t FTM_GetChlCounter (uint32_t instance, uint32_t chl)

获得FTM 通道计数值

参数
instance : 模块号
chl : 通道号

返回值
计数值

在文件 **ftm.c** 第 **219** 行定义.

uint32_t FTM_GetMoudlo (uint32_t instance)

Get FTM modulo.

参数
instance

返回值

modulo value

在文件 **ftm.c** 第 **245** 行定义.

```
void FTM_PWM_SetDuty ( uint32_t instance,
                       uint8_t  chl,
                       uint32_t duty
                     )
```

改变PWM占空比

参数

instance : 模块号
chl : 通道
pwmDuty : 占空比

返回值

None

在文件 **ftm.c** 第 **175** 行定义.

```
void FTM_SetMoudlo ( uint32_t instance,
                     uint32_t val
                   )
```

设置FTM主通道计数值

参数

instance : 模块号
val : value

返回值

None

在文件 **ftm.c** 第 **235** 行定义.

gpio.c 文件参考

浏览源代码.

函数

void	**GPIO_SetPinDir** (uint32_t instance, uint32_t pin, uint32_t dir)
uint32_t	**GPIO_Init** (uint32_t instance, uint32_t pin, GPIO_t mode)
void	**GPIO_PinWrite** (uint32_t instance, uint32_t pin, uint8_t data)
uint32_t	**GPIO_PinRead** (uint32_t instance, uint32_t pin)
void	**GPIO_PinToggle** (uint32_t instance, uint8_t pin)
uint32_t	**GPIO_ReadPort** (uint32_t instance)
void	**GPIO_WritePort** (uint32_t instance, uint32_t data)
int	**GPIO_SetIntMode** (uint32_t instance, uint32_t pin, GPIO_Int_t mode, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.5.28

在文件 **gpio.c** 中定义.

函数说明

uint32_t GPIO_Init (uint32_t instance,
uint32_t pin,
GPIO_t mode
)

初始化配置指定引脚的工作状态

注解
完成初始化一个引脚配置

参数
instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31
mode : 引脚操作模式

- kGPIO_IFT : 悬浮输入
- kGPIO_IPD : 下拉输入

- kGPIO_IPU : 上拉输入
- kGPIO_OOD : 开漏输出
- kGPIO_OPP : 推挽输出

返回值
None

在文件 gpio.c 第 87 行定义.

uint32_t GPIO_PinRead (uint32_t instance, uint32_t pin)

读取指定引脚的状态

注解
无

参数
instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31

返回值
返回引脚状态0或1

在文件 gpio.c 第 151 行定义.

void GPIO_PinToggle (uint32_t instance, uint8_t pin)

翻转引脚的状态

注解
None

参数
instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31

返回值
None

在文件 gpio.c 第 164 行定义.

void GPIO_PinWrite (uint32_t instance, uint32_t pin, uint8_t data

)

指定引脚输出状态

注解
设置指定引脚输出为0或1

参数

instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31
data : 引脚输出
- 0 : 输出0
- 1 : 输出1

返回值
None

在文件 **gpio.c** 第 **137** 行定义.

uint32_t GPIO_ReadPort (uint32_t instance)

读取引脚端口的状态

注解
None

参数

instance
- HW_GPIOx : GPIO端口号x

返回值
返回一个32位端口的数据

在文件 **gpio.c** 第 **176** 行定义.

int GPIO_SetIntMode (uint32_t instance, uint32_t pin, GPIO_Int_t mode, bool val)

设置引脚中断模式

注解
None

参数

instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31
mode : 32位的数据

返回值
None

在文件 **gpio.c** 第 **208** 行定义.

---

void GPIO_SetPinDir (uint32_t instance,
uint32_t pin,
uint32_t dir
)

设置引脚的输入输出方向

注解
控制引脚是输出还是输入

参数
instance
- HW_GPIOx : GPIO端口号x

pin : 引脚号码：0-31
dir : 方向选择参数

- 1 : 输入
- 0 : 输出

返回值
None

在文件 **gpio.c** 第 **67** 行定义.

---

void GPIO_WritePort (uint32_t instance,
uint32_t data
)

修改端口的输出状态

注解
None

参数
instance
- HW_GPIOx : GPIO端口号x

data : 32位的数据

返回值
None

在文件 **gpio.c** 第 **189** 行定义.

i2c.c 文件参考

浏览源代码.

函数

uint32_t	**I2C_Init** (uint32_t MAP, uint32_t baudrate)
uint8_t	**I2C_GetByte** (uint32_t instance)
uint8_t	**I2C_SendByte** (uint32_t instance, uint8_t data)
uint32_t	**I2C_BurstRead** (uint32_t instance, uint8_t addr, uint32_t regAddr, uint32_t regLen, uint8_t *buf, uint32_t len)
uint32_t	**I2C_InitEx** (uint32_t port, uint32_t sda_pin, uint32_t scl_pin, uint32_t baudrate)
uint32_t	**SCCB_ReadReg** (uint32_t instance, uint8_t addr, uint8_t regAddr, uint8_t *buf)
uint32_t	**SCCB_WriteReg** (uint32_t instance, uint8_t addr, uint8_t regAddr, uint8_t data)
uint32_t	**I2C_Probe** (uint32_t instance, uint8_t addr)
uint32_t	**I2C_BurstWrite** (uint32_t instance, uint8_t addr, uint32_t regAddr, uint32_t regLen, uint8_t *buf, uint32_t len)
uint32_t	**I2C_ReadReg** (uint32_t instance, uint8_t addr, uint8_t regAddr, uint8_t *buf)
uint32_t	**I2C_WriteReg** (uint32_t instance, uint8_t addr, uint8_t regAddr, uint8_t data)
void	**I2C_Scan** (uint32_t instance)

详细描述

作者
YANDLD

版本
V3.0

日期
2016.6.7

在文件 **i2c.c** 中定义.

函数说明

```
uint32_t I2C_BurstRead ( uint32_t  instance,
                         uint8_t   addr,
                         uint32_t  regAddr,
                         uint32_t  regLen,
                         uint8_t *  buf,
                         uint32_t  len
                       )
```

I2C连续性读取多字节

注解

None

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
regLen 寄存器数量
buf 准备发送的数据地址指针
len 数据长度

返回值

CH_OK：成功；CH_ERR：失败

注解

None

参数

instance:无意义

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
regLen 寄存器数量
buf 准备发送的数据地址指针
len 数据长度

返回值

0：成功；其它：失败

在文件 **i2c.c** 第 **320** 行定义.

```
uint32_t I2C_BurstWrite ( uint32_t  instance,
                          uint8_t   addr,
                          uint32_t  regAddr,
                          uint32_t  regLen,
                          uint8_t *  buf,
                          uint32_t  len
                        )
```

I2C连续性写入多字节

注解

None

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块

- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
regLen 寄存器数量
buf 准备发送的数据地址指针
len 数据长度

返回值
0：成功；其它：失败

在文件 **i2c.c** 第 **1170** 行定义.

static uint8_t I2C_GetByte (uint32_t instance)

读取I2C接收的数据

I2C读取一字节数据

注解
None

参数
instance
- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

返回值
返回I2C接收的一字节数据

注解
None

参数
instance:无意义
- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

返回值
接收到的数据

在文件 **i2c.c** 第 **236** 行定义.

uint32_t I2C_Init (uint32_t MAP, uint32_t baudrate)

初始化配置I2C

注解
None

参数

MAP:引脚位图，详见I2C.h

baudRate_Hz 通信速度

返回值

0:I2C0模块；1:I2C1模块；2:I2C2模块

在文件 **i2c.c** 第 **203** 行定义.

```
uint32_t I2C_InitEx ( uint32_t port,
                      uint32_t sda_pin,
                      uint32_t scl_pin,
                      uint32_t baudrate
                    )
```

I2C通信引脚初始化

注解

None

参数

port:引脚端口

sda_pin 数据引脚

scl_pin 时钟引脚

baudrate 通信速度

返回值

None

在文件 **i2c.c** 第 **758** 行定义.

```
uint32_t I2C_Probe ( uint32_t instance,
                     uint8_t  addr
                   )
```

探测I2C总线设备

注解

通用程序接口

检测目标设备是否可用

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址

返回值

0：成功；其它：错误

在文件 i2c.c 第 1143 行定义.

uint32_t I2C_ReadReg (uint32_t instance,
uint8_t addr,
uint8_t regAddr,
uint8_t * buf
)

读取单个寄存器数值

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
buf 读取的数据地址指针

返回值

0：成功；其它：错误

在文件 i2c.c 第 1207 行定义.

void I2C_Scan (uint32_t instance)

I2C总线浏览测试

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

注解

串口发送浏览到的设备ID

在文件 i2c.c 第 1236 行定义.

static bool I2C_SendByte (uint32_t instance,
uint8_t data
)

I2C发送一字节数据

注解

None

参数

instance

- HW_I2C0 : I2C0模块

- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

data 准备发送的数据

返回值

CH_OK：成功；**CH_ERR**：失败

注解

None

参数

instance:无意义

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

data 准备发送的数据

返回值

CH_OK：成功；**CH_ERR**：失败

在文件 **i2c.c** 第 **290** 行定义.

```
uint32_t I2C_WriteReg ( uint32_t  instance,
                        uint8_t   addr,
                        uint8_t   regAddr,
                        uint8_t   data
                      )
```

写单个寄存器

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址

regAddr 寄存器地址

data 准备写入的数据

返回值

0：成功；其它：错误

在文件 **i2c.c** 第 **1223** 行定义.

```
uint32_t SCCB_ReadReg ( uint32_t  instance,
                        uint8_t   addr,
                        uint8_t   regAddr,
                        uint8_t * buf
                      )
```

读取单个寄存器数值

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
buf 读取的数据地址指针

返回值

0：成功；其它：错误

在文件 **i2c.c** 第 **1061** 行定义.

```
uint32_t SCCB_WriteReg ( uint32_t instance,
                         uint8_t  addr,
                         uint8_t  regAddr,
                         uint8_t  data
                       )
```

写单个寄存器

参数

instance

- HW_I2C0 : I2C0模块
- HW_I2C1 : I2C1模块
- HW_I2C1 : I2C1模块

addr 目标设备地址
regAddr 寄存器地址
data 准备写入的数据

返回值

0：成功；其它：错误

在文件 **i2c.c** 第 **1105** 行定义.

lpit.c 文件参考

浏览源代码.

详细描述

作者
YANDLD

版本
V3.0.0

日期
2015.11.21

在文件 **lpit.c** 中定义.

lptmr.c 文件参考

浏览源代码.

函数

void	**LPTMR_TC_Init** (uint32_t instance, uint32_t ms)
void	**LPTMR_SetIntMode** (uint32_t instance, bool status)
void	**LPTMR_PC_Init** (uint32_t MAP, uint32_t polarity, uint32_t overFlowValue)
uint32_t	**LPTMR_ReadCounter** (uint32_t instance)
void	**LPTMR_SetTime** (uint32_t instance, uint32_t ms)
void	**LPTMR_ResetCounter** (uint32_t instance)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.08

在文件 **lptmr.c** 中定义.

函数说明

void LPTMR_PC_Init (uint32_t MAP,
uint32_t polarity,
uint32_t overFlowValue
)

初始化配置LPTMR模块在脉冲计数模式

注解
None

参数
MAP ：引脚位图信息，详见lptmr.h
polarity
- 0 : 下降沿脉冲计数
- 1 : 上升沿脉冲计数

overFlowValue：设置计数器计数最大值，以便设计中断

返回值
None

在文件 **lptmr.c** 第 **135** 行定义.

uint32_t LPTMR_ReadCounter (uint32_t instance)

读取LPTMR计数器计数值

注解

None

参数

instance

- HW_LPTMR0 : LPTMR0模块
- HW_LPTMR1 : LPTMR1模块

返回值

返回计数器的计数值

在文件 lptmr.c 第 187 行定义.

void LPTMR_ResetCounter (uint32_t instance)

重设LPTMR计时器时间

注解

清除计数器

参数

instance

- HW_LPTMR0 : LPTMR0模块
- HW_LPTMR1 : LPTMR1模块

返回值

None

在文件 lptmr.c 第 221 行定义.

void LPTMR_SetIntMode (uint32_t instance, bool status)

设置LPTMR中断模式

注解

None

参数

instance

- HW_LPTMR0 : LPTMR0模块
- HW_LPTMR1 : LPTMR1模块

status

- true : 开启中断
- flase : 关闭中断

返回值
None

在文件 **lptmr.c** 第 **114** 行定义.

---

void LPTMR_SetTime (uint32_t instance,
uint32_t ms
)

设置LPTMR计时器时间

注解
None

参数
instance
- HW_LPTMR0 : LPTMR0模块
- HW_LPTMR1 : LPTMR1模块

ms：时间 ms

返回值
None

在文件 **lptmr.c** 第 **205** 行定义.

---

void LPTMR_TC_Init (uint32_t instance,
uint32_t ms
)

初始化配置LPTMR工作在计时器模式

注解
计时器单位是ms

参数
instance
- HW_LPTMR0 : LPTMR0模块
- HW_LPTMR1 : LPTMR1模块

ms 计时时间

返回值
None

在文件 **lptmr.c** 第 **74** 行定义.

lpuart.c 文件参考

浏览源代码.

函数

uint32_t	**LPUART_SetClock** (uint32_t instance, uint32_t opt)
void	**LPUART_SetBaudRate** (uint32_t instance, uint32_t baud)
uint32_t	**LPUART_DeInit** (uint32_t instance)
uint32_t	**LPUART_Init** (uint32_t MAP, uint32_t baudrate)
void	**LPUART_PutChar** (uint32_t instance, uint8_t ch)
uint32_t	**LPUART_GetChar** (uint32_t instance, uint8_t *ch)
uint32_t	**LPUART_SetIntMode** (uint32_t instance, LPUART_Int_t mode, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.6.13

在文件 **lpuart.c** 中定义.

函数说明

uint32_t LPUART_DeInit (uint32_t instance)

删除串口原始配置

注解
None

参数
instance
- HW_UARTx : UART端口号0~5

返回值
删除完成

在文件 **lpuart.c** 第 **211** 行定义.

uint32_t LPUART_GetChar (uint32_t instance, uint8_t * ch)

串口接收一个字符

注解
None

参数
instance
- HW_UARTx : UART端口号0~5

ch 获得字符

返回值
返回接收结果，**CH_OK**：成功；**CH_ERR**：错误

在文件 **lpuart.c** 第 **271** 行定义.

---

uint32_t LPUART_Init (uint32_t MAP,
uint32_t baudrate
)

初始化配置串口

注解
None

参数
MAP:引脚地图，详见**lpuart.h**文件
baudrate : 通信波特率

返回值
串口端口号

在文件 **lpuart.c** 第 **230** 行定义.

---

void LPUART_PutChar (uint32_t instance,
uint8_t ch
)

串口发送一个字符

注解
None

参数
instance
- HW_UARTx : UART端口号0~5

ch 需要发送的字符

返回值
None

在文件 **lpuart.c** 第 **257** 行定义.

void LPUART_SetBaudRate (uint32_t instance,
uint32_t baud
)

设置串口波特率

注解
None

参数
instance
- HW_UARTx : UART端口号0~5

baud : 通信波特率

返回值
None

在文件 **lpuart.c** 第 **172** 行定义.

uint32_t LPUART_SetClock (uint32_t instance,
uint32_t opt
)

设置串口时钟源

注解
None

参数
instance
- HW_UARTx : UART端口号0-5

opt : 时钟源

返回值
时钟速度

在文件 **lpuart.c** 第 **109** 行定义.

uint32_t LPUART_SetIntMode (uint32_t instance,
LPUART_Int_t mode,
bool val
)

设置串口中断模式

注解
None

参数
instance
- HW_UARTx : UART端口号0~5

mode

- kUART_IntTx : UART-中断模式发送
- kUART_IntRx : UART-中断模式接收
- kUART_IntIdleLine : UART-IDLE模式接收

val :

- 0 : 开启中断模式
- 1 : 关闭中断模式

返回值

返回配置结果

在文件 **lpuart.c** 第 **303** 行定义.

mcg.c

```
    1
   10 #include "common.h"
   11 #include "mcg.h"
   12
   13 #if defined(MCG) && defined(MCG_C6_PLLS_MASK)
   14
   15 #if !defined(SIM_CLKDIV1_OUTDIV3)
   16     #define SIM_CLKDIV1_OUTDIV3(x)   (x & 0x00)
   17 #endif
   18
   19 #if !defined(SIM_CLKDIV1_OUTDIV2)
   20     #define SIM_CLKDIV1_OUTDIV2(x)   (x & 0x00)
   21 #endif
   22
   36 uint32_t ClockSetup(uint32_t opt)
   37 {
   38     static bool isInitialized = false;
   39
   40     if(isInitialized == true)
   41     {
   42         return CH_ERR;
   43     }
   44
   45     SIM->CLKDIV1 = 0xFFFFFFFF;
   46     if((opt == IRC_96M) || (opt == IRC_48M))
   47     {
   48         SIM->SOPT2 &= ~SIM_SOPT2_PLLFLLSEL_MASK;
   49         #if defined(SIM_SOPT1_OSC32KSEL_MASK)
   50         SIM->SOPT1 |= SIM_SOPT1_OSC32KSEL(0x03);
   51         #endif
   52
   53         /* Switch to FEI Mode */
   54         MCG->C1 = MCG_C1_CLKS(0x00) | MCG_C1_FRDIV(0x00) |
      MCG_C1_IREFS_MASK | MCG_C1_IRCLKEN_MASK | MCG_C1_IREFSTEN_MASK;
   55         MCG->C4 &= ~(MCG_C4_DMX32_MASK | MCG_C4_DRST_DRS_MASK);
   56         switch(opt)
   57         {
   58             case IRC_48M:
   59                 MCG->C4 |= (MCG_C4_DMX32_MASK |
      MCG_C4_DRST_DRS(0x01));
   60                 SIM->CLKDIV1 = SIM_CLKDIV1_OUTDIV1(0x00) |
      SIM_CLKDIV1_OUTDIV2(0x00) | SIM_CLKDIV1_OUTDIV3(0x00) |
      SIM_CLKDIV1_OUTDIV4(0x01);
   61                 SystemCoreClock = 48*1000*1000;
   62                 break;
   63             case IRC_96M:
   64                 MCG->C4 |= (MCG_C4_DMX32_MASK |
      MCG_C4_DRST_DRS(0x03));
   65                 SIM->CLKDIV1 = SIM_CLKDIV1_OUTDIV1(0x00) |
      SIM_CLKDIV1_OUTDIV2(0x01) | SIM_CLKDIV1_OUTDIV3(0x01) |
      SIM_CLKDIV1_OUTDIV4(0x03);
   66                 SystemCoreClock = 96*1000*1000;
   67                 break;
   68             default:
   69                 LIB_TRACE("Unsupported ClockSetup");
   70                 break;
   71         }
   72
   73         /* Check that the source of the FLL reference clock is
      the internal reference clock. */
   74         while((MCG->S & MCG_S_IREFST_MASK) == 0u);
   75         MCG->C6 &= ~MCG_C6_PLLS_MASK;
   76         /* Wait until output of the FLL is selected */
   77         while((MCG->S & 0x0Cu) != 0x00u);
   78     }
   79     else
```

```
80      {
81          /* FEI - FBE - PBE - PEE */
82          SIM->SOPT2 |= (SIM_SOPT2_PLLFLLSEL_MASK & 0x01);
83
84          /* config OSC */
85          #if defined(OSC)
86          OSC->CR = OSC_CR_ERCLKEN_MASK | OSC_CR_EREFSTEN_MASK;
87          #endif
88          SIM->SOPT7 = 0; /* select OSC0 as MCG input clock */
89          #if defined(MCG_C2_RANGE0_MASK)
90          MCG->C2 = MCG_C2_RANGE0(3) | MCG_C2_EREFS0_MASK;
91          #else
92          MCG->C2 = MCG_C2_RANGE(3) | MCG_C2_EREFS_MASK;
93          #endif
94          MCG->C1 = MCG_C1_CLKS(0x02); /* OSC as output clk */
95          while((MCG->S & MCG_S_OSCINIT0_MASK) == 0u);
96
97          /* if there is a PLL diver, set it to 1 */
98          #if defined(SIM_CLKDIV3_PLLFLLFRAC_MASK)
99          SIM->CLKDIV3 = 0;
100         #endif
101
102         /* config PLL */
103         switch(opt)
104         {
105             case EX50M_120M:
106                 MCG->C5 = (uint8_t)MCG_C5_PRDIV0(14);
107                 MCG->C6 = (uint8_t)(0x40u | MCG_C6_VDIV0(36-24));
108                 SystemCoreClock = 120*1000*1000;
109                 break;
110             case EX48M_96M:
111                 MCG->C5 = (uint8_t)MCG_C5_PRDIV0(11);
112                 MCG->C6 = (uint8_t)(0x40u | MCG_C6_VDIV0(24-24));
113                 SystemCoreClock = 96*1000*1000;
114                 break;
115             case EX48M_120M:
116                 MCG->C5 = (uint8_t)MCG_C5_PRDIV0(11);
117                 MCG->C6 = (uint8_t)(0x40u | MCG_C6_VDIV0(30-24));
118                 SystemCoreClock = 120*1000*1000;
119                 break;
120             case EX12M_120M:
121                 MCG->C5 = (uint8_t)MCG_C5_PRDIV0(1);
122                 MCG->C6 = (uint8_t)(0x40u | MCG_C6_VDIV0(40-16));
123                 SystemCoreClock = 120*1000*1000;
124                 break;
125         }
126
127         while((MCG->S & MCG_S_PLLST_MASK) == 0u);
128         while((MCG->S & MCG_S_LOCK0_MASK) == 0u);
129
130         /* select PLL as MCGOutClock */
131         MCG->C1 = MCG_C1_CLKS(0x00);
132         while((MCG->S & 0x0Cu) != 0x0Cu);                    /*
    PLL is selected */
133
134         SIM->CLKDIV1 =
    (SIM_CLKDIV1_OUTDIV1(0)|SIM_CLKDIV1_OUTDIV2(1)|SIM_CLKDIV1_OUTDIV3(
    1)|SIM_CLKDIV1_OUTDIV4(4));
135     }
136
137     isInitialized = true;
138     return CH_OK;
139 }
140
141 #endif
142
143
144
```

pit.c 文件参考

浏览源代码.

函数

void	**PIT_Init** (uint32_t chl, uint32_t us)
void	**PIT_SetTime** (uint32_t chl, uint32_t us)
uint32_t	**PIT_GetTime** (uint32_t chl)
uint32_t	**PIT_GetValue** (uint32_t chl)
void	**PIT_SetValue** (uint8_t chl, uint32_t val)
uint32_t	**PIT_SetIntMode** (uint32_t chl, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.05.31

在文件 **pit.c** 中定义.

函数说明

uint32_t PIT_GetTime (uint32_t chl)

获得PIT模块指定通道的计时时间

注解
None

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

返回值
此时计数器中的时间单位为**us**

在文件 **pit.c** 第 **88** 行定义.

uint32_t PIT_GetValue (uint32_t chl)

获得PIT模块指定通道的数值

注解
None

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

返回值
此时计数器中的数值

在文件 pit.c 第 101 行定义.

void PIT_Init (uint32_t chl,
uint32_t us
)

初始化配置PIT模块

注解
设置PIT通道的周期定时功能

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

us ：周期时间间隔

返回值
None

在文件 pit.c 第 53 行定义.

uint32_t PIT_SetIntMode (uint32_t chl,
bool val
)

设置PIT模块指定通道的中断状态

注解
None

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

val :

- 0 : 关闭中断
- 1 : 开启中断

在文件 pit.c 第 131 行定义.

void PIT_SetTime (uint32_t chl, uint32_t us)

设置PIT模块指定通道的时间

注解
设置PIT通道的周期时间间隔

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

us : 周期时间间隔

返回值
None

在文件 pit.c 第 73 行定义.

void PIT_SetValue (uint8_t chl, uint32_t val)

设置PIT模块指定通道的数值

注解
None

参数
chl
- HW_PIT_CH0 : PIT模块的0通道
- HW_PIT_CH1 : PIT模块的1通道

val :设置计数器的数值

在文件 pit.c 第 114 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_t *datetime)
bool	**RTC_IsTimeValid** (void)
void	**RTC_SetAlarm** (RTC_DateTime_t *datetime)
void	**RTC_GetAlarm** (RTC_DateTime_t *datetime)
void	**RTC_SetCompensation** (uint32_t compensationInterval, uint32_t timeCompensation)
void	**RTC_Init** (void)
uint32_t	**RTC_GetCounter** (void)
uint32_t	**RTC_GetTAR** (void)
void	**RTC_SetTime** (RTC_DateTime_t *datetime)
void	**RTC_SetTSR** (uint32_t val)
void	**RTC_SetIntMode** (RTC_Int_t mode, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.07

在文件 **rtc.c** 中定义.

函数说明

void RTC_GetAlarm (RTC_DateTime_t * datetime)

获得闹钟时间

参数
datetime 时间戳结构体

返回值
None

在文件 **rtc.c** 第 **217** 行定义.

uint32_t RTC_GetCounter (void)

获得TSR值

返回值

返回计时器数值

在文件 **rtc.c** 第 **287** 行定义.

uint32_t RTC_GetTAR (void)

获得Time Alarm值

返回值

返回闹钟计时器数值

在文件 **rtc.c** 第 **296** 行定义.

void RTC_GetTime (RTC_DateTime_t * datetime)

获得RTC的时间

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

参数

datetime 返回计算出来的年月日等信息结构体

返回值

None

在文件 **rtc.c** 第 **169** 行定义.

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

由年月日计算出周数

参数

year 年
month 月
days 日

返回值

返回计算出来的周期数

在文件 **rtc.c** 第 **52** 行定义.

void RTC_Init (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 **259** 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；其它：无效

在文件 **rtc.c** 第 **191** 行定义.

void RTC_SetAlarm (RTC_DateTime_t * datetime)

设置闹钟时间

参数

datetime 时间戳结构体

返回值

None

在文件 **rtc.c** 第 **205** 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 **247** 行定义.

void RTC_SetIntMode (RTC_Int_t mode, bool val)

设置RTC中断功能

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

参数
config 配置中断类型

- kRTC_IntSecond 秒中断
- kRTC_IntAlarm 闹钟中断

status 是否使能RTC中断

- false 关闭中断
- true 打开中断

返回值
None

在文件 **rtc.c** 第 **351** 行定义.

void RTC_SetTime (RTC_DateTime_t * datetime)

设置RTC的时间

参数
datetime 指向时间的结构体指针

返回值
None

在文件 **rtc.c** 第 **306** 行定义.

void RTC_SetTSR (uint32_t val)

设置RTC的Time Seconds Register

参数

val 总秒计数器

返回值

None

在文件 **rtc.c** 第 **325** 行定义.

scg.c 文件参考

浏览源代码.

详细描述

作者
YANDLD

版本
V3.0

日期
2016.05.27

在文件 **scg.c** 中定义.

sdhc.c 文件参考

浏览源代码.

函数

uint32_t	**SDHC_StatusWait** (uint32_t mask)
void	**SDHC_SetSDClk** (uint32_t instance, uint32_t sdClk)
uint32_t	**SDHC_SendCmd** (SDHC_Cmd_t *cmd)
uint32_t	**SD_CardInit** (uint32_t instance)
uint32_t	**SDHC_Init** (uint32_t MAP, uint32_t sdClk)
uint32_t	**SDHC_ReadBlock** (uint32_t sector, uint8_t *buf, uint32_t cnt)
uint32_t	**SDHC_WriteBlock** (uint32_t sector, uint8_t *buf, uint32_t cnt)
uint32_t	**SD_ReadSingleBlock** (uint32_t sector, uint8_t *buf)
uint32_t	**SD_WriteSingleBlock** (uint32_t sector, uint8_t *buf)
uint32_t	**SD_ReadMultiBlock** (uint32_t sector, uint8_t *buf, uint32_t cnt)
uint32_t	**SD_WriteMultiBlock** (uint32_t sector, uint8_t *buf, uint32_t cnt)

详细描述

作者
YANDLD

版本
V3.0

日期
2016.2.13

在文件 **sdhc.c** 中定义.

函数说明

uint32_t SD_CardInit (uint32_t instance)

SDHC initialize card.

注解
internal function

参数
[in] **cmd** 指令指针

返回值
0 CH_OK
other error code

在文件 **sdhc.c** 第 **295** 行定义.

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

SD_ReadMultiBlock legcy support.

注解
this function is same as SDHC_ReadBlock(...)

在文件 sdhc.c 第 693 行定义.

uint32_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 CH_OK
other error code

在文件 sdhc.c 第 631 行定义.

uint32_t SD_WriteMultiBlock (uint32_t sector,
uint8_t * buf,
uint32_t cnt
)

SD_WriteMultiBlock legcy support.

注解
this function is same as SDHC_WriteBlock(...)

在文件 sdhc.c 第 702 行定义.

uint32_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 CH_OK

other error code

在文件 **sdhc.c** 第 **643** 行定义.

---

uint32_t SDHC_Init (uint32_t MAP,
uint32_t sdClk
)

SDHC initialize.

参数

[in] **MAP**

[in] **sd** clock

返回值

CH_OK successfull

other failed

在文件 **sdhc.c** 第 **451** 行定义.

---

uint32_t SDHC_ReadBlock (uint32_t sector,
uint8_t * buf,
uint32_t cnt
)

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 CH_OK

other error code

在文件 **sdhc.c** 第 **497** 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 CH_OK

other error code

在文件 sdhc.c 第 231 行定义.

void SDHC_SetSDClk (uint32_t instance, uint32_t sdClk)

Set SDHC baud rate.

参数

[in] clock 时钟频率
[in] baudrate 波特率设置

返回

None

在文件 sdhc.c 第 183 行定义.

uint32_t SDHC_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 sdhc.c 第 122 行定义.

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

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 CH_OK
other error code

在文件 sdhc.c 第 552 行定义.

sdramc.c 文件参考

浏览源代码.

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.2.16

在文件 **sdramc.c** 中定义.

spi.c 文件参考

浏览源代码.

函数

void	**SPI_Config** (uint32_t instance, uint32_t format)
uint32_t	**SPI_Init** (uint32_t MAP, uint32_t baudrate)
uint32_t	**SPI_ReadWrite** (uint32_t instance, uint32_t data)
uint32_t	**SPI_SetIntMode** (uint32_t instance, SPI_Int_t mode, bool val)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.12

在文件 **spi.c** 中定义.

函数说明

void SPI_Config (uint32_t instance,
uint32_t format
)

配置SPI模块

注解
None

参数
instance:无意义
- HW_SPI0 : SPI0模块
- HW_SPI1 : SPI1模块
- HW_SPI2 : SPI2模块

MAP:引脚位图，详见spi.h文件
format 通信格式

返回值
None

在文件 **spi.c** 第 **131** 行定义.

uint32_t SPI_Init (uint32_t MAP,

uint32_t baudrate
)

初始化配置SPI

注解
None

参数
MAP:引脚位图，详见spi.h文件
baudRate_Hz 通信速度

返回值
0:SPI0模块；1:SPI1模块；2:SPI2模块

在文件 spi.c 第 166 行定义.

uint32_t SPI_ReadWrite (uint32_t instance,
uint32_t data
)

SPI模块读写

注解
None

参数
instance:无意义
- HW_SPI0 : SPI0模块
- HW_SPI1 : SPI1模块
- HW_SPI2 : SPI2模块

data 写数据

返回值
读数据

在文件 spi.c 第 202 行定义.

uint32_t SPI_SetIntMode (uint32_t instance,
SPI_Int_t mode,
bool val
)

设置SPI中断模式

注解
None

参数
instance
- HW_SPI0 : SPI0模块
- HW_SPI1 : SPI1模块
- HW_SPI2 : SPI2模块

status

- true : 开启中断
- flase : 关闭中断

返回值

0:成功；其它：错误

在文件 **spi.c** 第 **207** 行定义.

tpm.c 文件参考

浏览源代码.

函数

uint32_t	**TPM_PWM_Init** (uint32_t MAP, TPM_t mode, uint32_t Hz)
void	**TPM_PWM_Invert** (uint32_t instance, uint8_t chl)
void	**TPM_PWM_SetDuty** (uint32_t instance, uint8_t chl, uint32_t duty)
void	**TPM_SetIntMode** (uint32_t instance, TPM_Int_t mode, bool val)
uint32_t	**TPM_GetChlCounter** (uint32_t instance, uint32_t chl)
void	**TPM_SetChlCounter** (uint32_t instance, uint32_t chl, uint32_t val)
void	**TPM_SetMoudlo** (uint32_t instance, uint32_t val)
uint32_t	**TPM_GetMoudlo** (uint32_t instance)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2016.06.12

在文件 **tpm.c** 中定义.

函数说明

uint32_t TPM_GetChlCounter (uint32_t instance,
uint32_t chl
)

获得TPM通道计数值

参数
instance
- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

chl : 通道0~5

返回值
计数值

在文件 **tpm.c** 第 **246** 行定义.

uint32_t TPM_GetMoudlo (uint32_t instance)

获得TPM主通道计数值

参数

instance

- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

返回值

计数器数值

在文件 **tpm.c** 第 **285** 行定义.

uint32_t TPM_PWM_Init (uint32_t MAP, TPM_t mode, uint32_t Hz)

初始化TPM模块为PWM模式

注解

默认输出占空比0%

参数

MAP 初始化信息，详见tpm.h文件
mode PWM模式，详见tpm.h文件
Hz 输出频率

返回值

CH_OK 成功 其他:失败

在文件 **tpm.c** 第 **93** 行定义.

void TPM_PWM_Invert (uint32_t instance, uint8_t chl)

改变PWM输出极性

注解

调转PWM输出极性

参数

instance

- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

chl : PWM通道

返回值

None

在文件 tpm.c 第 171 行定义.

```
void TPM_PWM_SetDuty ( uint32_t instance,
                       uint8_t  chl,
                       uint32_t duty
                     )
```

改变PWM占空比

注解
None

参数
instance
- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

chl : PWM通道
duty : 占空比0~10000 对应 0-100%

返回值
None

在文件 tpm.c 第 189 行定义.

```
void TPM_SetChlCounter ( uint32_t instance,
                         uint32_t chl,
                         uint32_t val
                       )
```

设置TPM通道计数值

参数
instance
- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

chl : 通道0~5
val : 计数器数值

返回值
None

在文件 tpm.c 第 260 行定义.

```
void TPM_SetIntMode ( uint32_t    instance,
                      TPM_Int_t mode,
                      bool        val
                    )
```

改变PWM占空比

注解
None

参数
instance
- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

mode : 中断模式，详见h文件
duty : 占空比0~10000
val : 1：开启；0关闭

返回值
None

在文件 **tpm.c** 第 **212** 行定义.

void TPM_SetMoudlo (uint32_t instance,
uint32_t val
)

设置TPM主通道计数值

参数
instance
- HW_TPM0 : TPM0模块
- HW_TPM1 : TPM1模块

val : value

返回值
None

在文件 **tpm.c** 第 **273** 行定义.

uart.c 文件参考

浏览源代码.

函数

uint32_t	**UART_SetClock** (uint32_t instance, uint32_t opt)
void	**UART_SetBaudRate** (uint32_t instance, uint32_t baud)
uint32_t	**UART_Init** (uint32_t MAP, uint32_t baudrate)
uint32_t	**UART_DeInit** (uint32_t MAP)
uint32_t	**UART_SetDMAMode** (uint32_t instance, UART_DMA_t mode, bool val)
uint32_t	**UART_SetIntMode** (uint32_t instance, UART_Int_t mode, bool val)
void	**UART_PutChar** (uint32_t instance, uint8_t ch)
uint32_t	**UART_GetChar** (uint32_t instance, uint8_t *ch)

详细描述

作者
YANDLD

版本
V3.0.0

日期
2015.6.21

在文件 **uart.c** 中定义.

函数说明

uint32_t UART_DeInit (uint32_t MAP)

反初始化UART

注解
None

参数
MAP:初始化信息,见**uart.h**

返回值
CH_OK 成功 其他:失败

在文件 **uart.c** 第 **221** 行定义.

uint32_t UART_GetChar (uint32_t instance, uint8_t * ch)

串口接收一个字符

注解

None

参数

instance

- HW_UARTx : UART端口号x ch: 接受到的字符地址

ch 获得字符

返回值

CH_OK 成功 其他:失败

在文件 **uart.c** 第 **354** 行定义.

```
uint32_t UART_Init ( uint32_t  MAP,
                     uint32_t  baudrate
                   )
```

初始化配置串口

注解

None

参数

MAP:初始化信息,见uart.h

baudrate : 通信波特率

返回值

通信波特率

在文件 **uart.c** 第 **196** 行定义.

```
void UART_PutChar ( uint32_t  instance,
                    uint8_t   ch
                  )
```

串口发送一个字符

注解

None

参数

instance

- HW_UARTx : UART端口号x

ch 需要发送的字符

返回值

None

在文件 **uart.c** 第 **338** 行定义.

```
void UART_SetBaudRate ( uint32_t instance,
                        uint32_t baud
                      )
```

设置串口波特率

注解
None

参数
instance
- HW_UARTx : UART端口号x

baud : 通信波特率

返回值
None

在文件 **uart.c** 第 **150** 行定义.

```
uint32_t UART_SetClock ( uint32_t instance,
                         uint32_t opt
                       )
```

设置串口时钟源

注解
None

参数
instance
- HW_UARTx : UART端口号x

opt : 时钟源

返回值
时钟速度

在文件 **uart.c** 第 **110** 行定义.

```
uint32_t UART_SetDMAMode ( uint32_t      instance,
                           UART_DMA_t    mode,
                           bool          val
                         )
```

设置串口DMA模式

注解
None

参数
instance
- HW_UARTx : UART端口号x

mode

- kUART_DMATx : UART-DMA 模式发送
- kUART_DMARx : UART-DMA 模式接收

val :

- 0 : 开启UART DMA请求
- 1 : 关闭UART DMA请求

返回值

CH_OK 成功 其他:失败

在文件 **uart.c** 第 **247** 行定义.

---

uint32_t UART_SetIntMode (uint32_t instance,
UART_Int_t mode,
bool val
)

设置串口中断模式

注解

None

参数

instance

- HW_UARTx : UART端口号x

mode

- kUART_IntTx : UART- 发送完成中断
- kUART_IntRx : UART- 接收完成中断
- kUART_IntIdleLine : UART-IDLE模式接收

val :

- true : 开启中断模式
- false : 关闭中断模式

返回值

CH_OK 成功 其他:失败

在文件 **uart.c** 第 **308** 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:

- _ -

- _flexio_pin_config : **flexio.c**
- _flexio_pin_polarity : **flexio.c**
- _flexio_shifter_buffer_type : **flexio.c**
- _flexio_shifter_input_source : **flexio.c**
- _flexio_shifter_mode : **flexio.c**
- _flexio_shifter_start_bit : **flexio.c**
- _flexio_shifter_stop_bit : **flexio.c**
- _flexio_shifter_timer_polarity : **flexio.c**
- _flexio_timer_decrement_source : **flexio.c**
- _flexio_timer_disable_condition : **flexio.c**
- _flexio_timer_enable_condition : **flexio.c**
- _flexio_timer_mode : **flexio.c**
- _flexio_timer_output : **flexio.c**
- _flexio_timer_reset_condition : **flexio.c**
- _flexio_timer_start_bit_condition : **flexio.c**
- _flexio_timer_stop_bit_condition : **flexio.c**
- _flexio_timer_trigger_polarity : **flexio.c**
- _flexio_timer_trigger_source : **flexio.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:

- a -

- ADC_Init() : **adc.c**
- ADC_SetChlMux() : **adc.c**
- ADC_SetIntMode() : **adc.c**
- ADC_SetTrigMode() : **adc.c**
- ADC_SoftRead() : **adc.c**
- ADC_SoftTrigger() : **adc.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 -

- DelayInit() : **common.c**
- DelayMs() : **common.c**
- DelayUs() : **common.c**
- DMA_ClearIntFlag() : **dma.c**
- DMA_GetDestAddr() : **dma.c**
- DMA_GetSrcAddr() : **dma.c**
- DMA_GetTransCnt() : **dma.c**
- DMA_Init() : **dma.c**
- DMA_IsTransDone() : **dma.c**
- DMA_SetDestAddr() : **dma.c**
- DMA_SetDestMod() : **dma.c**
- DMA_SetIntMode() : **dma.c**
- DMA_SetSrcAddr() : **dma.c**
- DMA_SetSrcMod() : **dma.c**
- DMA_SetTransCnt() : **dma.c**
- DMA_Start() : **dma.c**
- DMA_Stop() : **dma.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:

- e -

- ENET_Init() : **enet.c**
- ENET_PHY_Init() : **enet.c**
- ENET_PHY_Read() : **enet.c**
- ENET_PHY_Write() : **enet.c**
- ENET_ReceiveData() : **enet.c**
- ENET_SendData() : **enet.c**
- ENET_SetMacAddr() : **enet.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**
- GetResetCause() : **common.c**
- GetUID() : **common.c**
- GPIO_Init() : **gpio.c**
- GPIO_PinRead() : **gpio.c**
- GPIO_PinToggle() : **gpio.c**
- GPIO_PinWrite() : **gpio.c**
- GPIO_ReadPort() : **gpio.c**
- GPIO_SetIntMode() : **gpio.c**
- GPIO_SetPinDir() : **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:

- 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:

- k -

- kFLEXIO_TimerOutputOneNotAffectedByReset : **flexio.c**
- kFLEXIO_TimerOutputZeroAffectedByReset : **flexio.c**
- kFLEXIO_TimerOutputZeroNotAffectedByReset : **flexio.c**
- kFLEXIO_TimerResetNever : **flexio.c**
- kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput : **flexio.c**
- kFLEXIO_TimerResetOnTimerPinRisingEdge : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerBothEdge : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerRisingEdge : **flexio.c**
- kFLEXIO_TimerStartBitDisabled : **flexio.c**
- kFLEXIO_TimerStartBitEnabled : **flexio.c**
- kFLEXIO_TimerStopBitDisabled : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerCompare : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerCompareDisable : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerDisable : **flexio.c**
- kFLEXIO_TimerTriggerPolarityActiveHigh : **flexio.c**
- kFLEXIO_TimerTriggerPolarityActiveLow : **flexio.c**
- kFLEXIO_TimerTriggerSourceExternal : **flexio.c**
- kFLEXIO_TimerTriggerSourceInternal : **flexio.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:

- l -

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 -

- 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 -

- PIT_GetTime() : **pit.c**
- PIT_GetValue() : **pit.c**
- PIT_Init() : **pit.c**
- PIT_SetIntMode() : **pit.c**
- PIT_SetTime() : **pit.c**
- PIT_SetValue() : **pit.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 -

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 -

- TPM_GetChlCounter() : **tpm.c**
- TPM_GetMoudlo() : **tpm.c**
- TPM_PWM_Init() : **tpm.c**
- TPM_PWM_Invert() : **tpm.c**
- TPM_PWM_SetDuty() : **tpm.c**
- TPM_SetChlCounter() : **tpm.c**
- TPM_SetIntMode() : **tpm.c**
- TPM_SetMoudlo() : **tpm.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 -

- UART_DeInit() : **uart.c**
- UART_GetChar() : **uart.c**
- UART_Init() : **uart.c**
- UART_PutChar() : **uart.c**
- UART_SetBaudRate() : **uart.c**
- UART_SetClock() : **uart.c**
- UART_SetDMAMode() : **uart.c**
- UART_SetIntMode() : **uart.c**

- a -

- ADC_Init() : **adc.c**
- ADC_SetChlMux() : **adc.c**
- ADC_SetIntMode() : **adc.c**
- ADC_SetTrigMode() : **adc.c**
- ADC_SoftRead() : **adc.c**
- ADC_SoftTrigger() : **adc.c**

- d -

- DelayInit() : **common.c**
- DelayMs() : **common.c**
- DelayUs() : **common.c**
- DMA_ClearIntFlag() : **dma.c**
- DMA_GetDestAddr() : **dma.c**
- DMA_GetSrcAddr() : **dma.c**
- DMA_GetTransCnt() : **dma.c**
- DMA_Init() : **dma.c**
- DMA_IsTransDone() : **dma.c**
- DMA_SetDestAddr() : **dma.c**
- DMA_SetDestMod() : **dma.c**
- DMA_SetIntMode() : **dma.c**
- DMA_SetSrcAddr() : **dma.c**
- DMA_SetSrcMod() : **dma.c**
- DMA_SetTransCnt() : **dma.c**
- DMA_Start() : **dma.c**
- DMA_Stop() : **dma.c**

- e -

- ENET_Init() : **enet.c**
- ENET_PHY_Init() : **enet.c**
- ENET_PHY_Read() : **enet.c**
- ENET_PHY_Write() : **enet.c**
- ENET_ReceiveData() : **enet.c**
- ENET_SendData() : **enet.c**
- ENET_SetMacAddr() : **enet.c**

- f -

- FLASH_EraseSector() : **flash.c**
- FLASH_GetSectorSize() : **flash.c**
- FLASH_Init() : **flash.c**
- FLASH_Test() : **flash.c**
- FLASH_WriteSector() : **flash.c**
- FLEXBUS_Init() : **flexbus.c**
- FLEXBUS_TimingConfig() : **flexbus.c**
- FLEXIO_Init() : **flexio.c**
- FLEXIO_PWM_Init() : **flexio.c**
- FLEXIO_PWM_Start() : **flexio.c**
- FLEXIO_PWM_Stop() : **flexio.c**
- FLEXIO_SetShifterConfig() : **flexio.c**
- FLEXIO_SetTimerConfig() : **flexio.c**
- FLEXIO_UART_GetChar() : **flexio.c**
- FLEXIO_UART_Init() : **flexio.c**

- FLEXIO_UART_PutChar() : **flexio.c**
- FTM_GetChlCounter() : **ftm.c**
- FTM_GetMoudlo() : **ftm.c**
- FTM_PWM_SetDuty() : **ftm.c**
- FTM_SetMoudlo() : **ftm.c**

- g -

- GetClock() : **common.c**
- GetResetCause() : **common.c**
- GetUID() : **common.c**
- GPIO_Init() : **gpio.c**
- GPIO_PinRead() : **gpio.c**
- GPIO_PinToggle() : **gpio.c**
- GPIO_PinWrite() : **gpio.c**
- GPIO_ReadPort() : **gpio.c**
- GPIO_SetIntMode() : **gpio.c**
- GPIO_SetPinDir() : **gpio.c**
- GPIO_WritePort() : **gpio.c**

- i -

- I2C_BurstRead() : **i2c.c**
- I2C_BurstWrite() : **i2c.c**
- I2C_GetByte() : **i2c.c**
- I2C_Init() : **i2c.c**
- I2C_InitEx() : **i2c.c**
- I2C_Probe() : **i2c.c**
- I2C_ReadReg() : **i2c.c**
- I2C_Scan() : **i2c.c**
- I2C_SendByte() : **i2c.c**
- I2C_WriteReg() : **i2c.c**

- l -

- LPTMR_PC_Init() : **lptmr.c**
- LPTMR_ReadCounter() : **lptmr.c**
- LPTMR_ResetCounter() : **lptmr.c**
- LPTMR_SetIntMode() : **lptmr.c**
- LPTMR_SetTime() : **lptmr.c**
- LPTMR_TC_Init() : **lptmr.c**
- LPUART_DeInit() : **lpuart.c**
- LPUART_GetChar() : **lpuart.c**
- LPUART_Init() : **lpuart.c**
- LPUART_PutChar() : **lpuart.c**
- LPUART_SetBaudRate() : **lpuart.c**
- LPUART_SetClock() : **lpuart.c**
- LPUART_SetIntMode() : **lpuart.c**

- n -

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

- p -

- PIT_GetTime() : **pit.c**
- PIT_GetValue() : **pit.c**
- PIT_Init() : **pit.c**
- PIT_SetIntMode() : **pit.c**

- PIT_SetTime() : **pit.c**
- PIT_SetValue() : **pit.c**

- r -

- RTC_GetAlarm() : **rtc.c**
- RTC_GetCounter() : **rtc.c**
- RTC_GetTAR() : **rtc.c**
- RTC_GetTime() : **rtc.c**
- RTC_GetWeek() : **rtc.c**
- RTC_Init() : **rtc.c**
- RTC_IsTimeValid() : **rtc.c**
- RTC_SetAlarm() : **rtc.c**
- RTC_SetCompensation() : **rtc.c**
- RTC_SetIntMode() : **rtc.c**
- RTC_SetTime() : **rtc.c**
- RTC_SetTSR() : **rtc.c**

- s -

- SCCB_ReadReg() : **i2c.c**
- SCCB_WriteReg() : **i2c.c**
- SD_CardInit() : **sdhc.c**
- SD_ReadMultiBlock() : **sdhc.c**
- SD_ReadSingleBlock() : **sdhc.c**
- SD_WriteMultiBlock() : **sdhc.c**
- SD_WriteSingleBlock() : **sdhc.c**
- SDHC_Init() : **sdhc.c**
- SDHC_ReadBlock() : **sdhc.c**
- SDHC_SendCmd() : **sdhc.c**
- SDHC_SetSDClk() : **sdhc.c**
- SDHC_StatusWait() : **sdhc.c**
- SDHC_WriteBlock() : **sdhc.c**
- SetPinMux() : **common.c**
- SetPinOpenDrain() : **common.c**
- SetPinPull() : **common.c**
- SetPowerMode() : **common.c**
- SPI_Config() : **spi.c**
- SPI_Init() : **spi.c**
- SPI_ReadWrite() : **spi.c**
- SPI_SetIntMode() : **spi.c**
- SystemSoftReset() : **common.c**
- SysTick_SetIntMode() : **common.c**
- SysTick_SetTime() : **common.c**

- t -

- TPM_GetChlCounter() : **tpm.c**
- TPM_GetMoudlo() : **tpm.c**
- TPM_PWM_Init() : **tpm.c**
- TPM_PWM_Invert() : **tpm.c**
- TPM_PWM_SetDuty() : **tpm.c**
- TPM_SetChlCounter() : **tpm.c**
- TPM_SetIntMode() : **tpm.c**
- TPM_SetMoudlo() : **tpm.c**

- u -

- UART_DeInit() : **uart.c**

- UART_GetChar() : **uart.c**
- UART_Init() : **uart.c**
- UART_PutChar() : **uart.c**
- UART_SetBaudRate() : **uart.c**
- UART_SetClock() : **uart.c**
- UART_SetDMAMode() : **uart.c**
- UART_SetIntMode() : **uart.c**

- flexio_config_t : **flexio.c**
- flexio_pin_config_t : **flexio.c**
- flexio_pin_polarity_t : **flexio.c**
- flexio_shifter_buffer_type_t : **flexio.c**
- flexio_shifter_config_t : **flexio.c**
- flexio_shifter_input_source_t : **flexio.c**
- flexio_shifter_mode_t : **flexio.c**
- flexio_shifter_start_bit_t : **flexio.c**
- flexio_shifter_stop_bit_t : **flexio.c**
- flexio_shifter_timer_polarity_t : **flexio.c**
- flexio_timer_config_t : **flexio.c**
- flexio_timer_decrement_source_t : **flexio.c**
- flexio_timer_disable_condition_t : **flexio.c**
- flexio_timer_enable_condition_t : **flexio.c**
- flexio_timer_mode_t : **flexio.c**
- flexio_timer_output_t : **flexio.c**
- flexio_timer_reset_condition_t : **flexio.c**
- flexio_timer_start_bit_condition_t : **flexio.c**
- flexio_timer_stop_bit_condition_t : **flexio.c**
- flexio_timer_trigger_polarity_t : **flexio.c**
- flexio_timer_trigger_source_t : **flexio.c**

- _flexio_pin_config : **flexio.c**
- _flexio_pin_polarity : **flexio.c**
- _flexio_shifter_buffer_type : **flexio.c**
- _flexio_shifter_input_source : **flexio.c**
- _flexio_shifter_mode : **flexio.c**
- _flexio_shifter_start_bit : **flexio.c**
- _flexio_shifter_stop_bit : **flexio.c**
- _flexio_shifter_timer_polarity : **flexio.c**
- _flexio_timer_decrement_source : **flexio.c**
- _flexio_timer_disable_condition : **flexio.c**
- _flexio_timer_enable_condition : **flexio.c**
- _flexio_timer_mode : **flexio.c**
- _flexio_timer_output : **flexio.c**
- _flexio_timer_reset_condition : **flexio.c**
- _flexio_timer_start_bit_condition : **flexio.c**
- _flexio_timer_stop_bit_condition : **flexio.c**
- _flexio_timer_trigger_polarity : **flexio.c**
- _flexio_timer_trigger_source : **flexio.c**

- k -

- kFLEXIO_TimerOutputOneNotAffectedByReset : **flexio.c**
- kFLEXIO_TimerOutputZeroAffectedByReset : **flexio.c**
- kFLEXIO_TimerOutputZeroNotAffectedByReset : **flexio.c**
- kFLEXIO_TimerResetNever : **flexio.c**
- kFLEXIO_TimerResetOnTimerPinEqualToTimerOutput : **flexio.c**
- kFLEXIO_TimerResetOnTimerPinRisingEdge : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerBothEdge : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerEqualToTimerOutput : **flexio.c**
- kFLEXIO_TimerResetOnTimerTriggerRisingEdge : **flexio.c**
- kFLEXIO_TimerStartBitDisabled : **flexio.c**
- kFLEXIO_TimerStartBitEnabled : **flexio.c**
- kFLEXIO_TimerStopBitDisabled : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerCompare : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerCompareDisable : **flexio.c**
- kFLEXIO_TimerStopBitEnableOnTimerDisable : **flexio.c**
- kFLEXIO_TimerTriggerPolarityActiveHigh : **flexio.c**
- kFLEXIO_TimerTriggerPolarityActiveLow : **flexio.c**
- kFLEXIO_TimerTriggerSourceExternal : **flexio.c**
- kFLEXIO_TimerTriggerSourceInternal : **flexio.c**