# GD32 MCU 外设固件库使用手册

——基于《GD32F10x_Firmware_Library_V1.0.0》固件库

# 目录

# 一、GPIO 模块

GPIO 模块固件库文件为 gd32f10x_gpio.c 和 gd32f10x_gpio.h，该 GPIO 固件库可用作多个用途，包括 GPIO 引脚设置、电平输出、读取电平输入、配置为复用功能、配置为外部中断功能、引脚锁定等。

## 1.1 GPIO 寄存器

GPIO 模块的寄存器定义在 gd32f10x.h 文件中，具体定义见代码清单 1.1.1。

代码清单 1.1.1 GPIO 模块寄存器定义

```
typedef struct
{
    __IO uint32_t CTLR1;
    __IO uint32_t CTLR2;
    __IO uint32_t DIR;
    __IO uint32_t DOR;
    __IO uint32_t BOR;
    __IO uint32_t BCR;
    __IO uint32_t LOCKR;
} GPIO_TypeDef;

typedef struct
{
    __IO uint32_t AFIO_EVR;
    __IO uint32_t AFIO_PCFR1;
    __IO uint32_t AFIO_ESSR1;
    __IO uint32_t AFIO_ESSR2;
    __IO uint32_t AFIO_ESSR3;
    __IO uint32_t AFIO_ESSR4;
    uint32_t RESERVED0;
    __IO uint32_t AFIO_PCFR2;
} AFIO_TypeDef;
```

GPIO 寄存器映射我们以 GPIOA 为例进行说明，具体见代码清单 1.2.2 所示。

代码清单 1.1.2 GPIOA 寄存器映射

```
#define GPIOA                   ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOA_BASE              (APB2PERIPH_BASE + 0x0800)
#define APB2PERIPH_BASE         (PERIPH_BASE + 0x10000)
#define PERIPH_BASE             ((uint32_t)0x40000000)
```

在代码清单 1.1.2 中，第 1 行代码将 GPIO 定义为存储在 GPIOA_BASE 地址中类型为 GPIO_TypeDef 的结构体数据，因此该结构体中的第一个变量 CTLR1 的地址为 (GPIOA_BASE+1)，其余的结构体变量在内存中依次排列。在该代码清单中，第 2 行代码为定义 GPIOA_BASE 地址，该地址为从 APB2PERIPH_BASE 地址增加 0x0800 的地址偏移量，第 3 行代码为定义 APB2PERIPH_BASE 地址，该地址为从 PERIPH_BASE 外设基地址增加 0x10000 的地址偏移量，第 4 行代码定义了 PERIPH_BASE 外设基地址，为 0x40000000。利

用以上代码可实现 GPIO 寄存器的地址映射，GD32 MCU 内的其余片内外设均采用该地址映射的方式，在其余章节中，仅贴出代码清单，详细说明将不再赘述。

## 1.2 GPIO 初始化结构体类型 GPIO_InitPara

GPIO 初始化结构体定义如代码清单 1.2.1 所示。

代码清单 1.2.1 GPIO 初始化结构体定义代码

```
typedef struct
{
    uint16_t GPIO_Pin;
    GPIO_SpeedPara GPIO_Speed;
    GPIO_ModePara GPIO_Mode;
}GPIO_InitPara;
```

GPIO_InitPara 结构体中定义了三个成员变量，具体描述如表 1.2.1.1 所示。

表 1.2.1.1 GPIO_InitPara 结构体成员变量描述表

| 类型 | 成员变量 | 功能描述 | 举例 |
|---|---|---|---|
| uint16_t | GPIO_Pin | 所需要配置的引脚 | GPIO_PIN_0 |
| GPIO_SpeedPara | GPIO_Speed | 引脚输出速度配置 | GPIO_SPEED_50MHZ |
| GPIO_ModePara | GPIO_Mode | 所选引脚的工作模式 | GPIO_MODE_AIN |

其中，GPIO_InitPara ->GPIO_Pin 成员变量可从 GPIO_pins_define 所定义的常量中获取，GPIO_pins_define 的常量定义如代码清单 1.2.2 所示。

代码清单 1.2.2 GPIO_pins_define 定义代码

```
#define GPIO_PIN_0                 ((uint16_t)0x0001)
#define GPIO_PIN_1                 ((uint16_t)0x0002)
#define GPIO_PIN_2                 ((uint16_t)0x0004)
#define GPIO_PIN_3                 ((uint16_t)0x0008)
#define GPIO_PIN_4                 ((uint16_t)0x0010)
#define GPIO_PIN_5                 ((uint16_t)0x0020)
#define GPIO_PIN_6                 ((uint16_t)0x0040)
#define GPIO_PIN_7                 ((uint16_t)0x0080)
#define GPIO_PIN_8                 ((uint16_t)0x0100)
#define GPIO_PIN_9                 ((uint16_t)0x0200)
#define GPIO_PIN_10                ((uint16_t)0x0400)
#define GPIO_PIN_11                ((uint16_t)0x0800)
#define GPIO_PIN_12                ((uint16_t)0x1000)
#define GPIO_PIN_13                ((uint16_t)0x2000)
#define GPIO_PIN_14                ((uint16_t)0x4000)
#define GPIO_PIN_15                ((uint16_t)0x8000)
#define GPIO_PIN_ALL               ((uint16_t)0xFFFF)
```

GPIO_InitPara ->GPIO_Speed 成员变量可从 GPIO_SpeedPara 共用体中获取，GPIO_SpeedPara 共用体定义如代码清单 1.2.3 所示

代码清单 1.2.3 GPIO_SpeedPara 共用体定义代码

```
typedef enum
{
    GPIO_SPEED_10MHZ = 1,
    GPIO_SPEED_2MHZ,
    GPIO_SPEED_50MHZ
}GPIO_SpeedPara;
```

GPIO_InitPara -> GPIO_Mode 成员变量可从 GPIO_ModePara 共用体中获取，GPIO_ModePara 共用体类型定义如代码清单 1.2.4 所示。

代码清单 1.2.4 GPIO_ModePara 共用体类型定义代码清单

```
typedef enum
{
    GPIO_MODE_AIN = 0x0,
    GPIO_MODE_IN_FLOATING = 0x04,
    GPIO_MODE_IPD = 0x28,
    GPIO_MODE_IPU = 0x48,
    GPIO_MODE_OUT_OD = 0x14,
    GPIO_MODE_OUT_PP = 0X10,
    GPIO_MODE_AF_OD = 0X1C,
    GPIO_MODE_AF_PP = 0X18
}GPIO_ModePara;
```

## 1.3 GPIO 库函数

GPIO 库函数列表如表 1.3.1 所示。

表 1.3.1 GPIO 库函数列表

| 函数名称 | 功能描述 |
|---|---|
| GPIO_AFDeInit | 将复用功能(remap, event control, EXTI configuration)设置为缺省值 |
| GPIO_DeInit | 将 GPIO 寄存器设置为缺省值 |
| GPIO_ETH_MediaInterfaceConfig | 将 GPIO 引脚配置为以太网多媒体接口(仅支持 GD32 互联型产品) |
| GPIO_EventOutputConfig | 将 GPIO 引脚配置为事件输出功能 |
| GPIO_EXTILineConfig | 将 GPIO 引脚配置为 EXTI 外部中断功能 |
| GPIO_Init | GPIO 模块初始化 |
| GPIO_ParaInit | 初始化 GPIO 初始化结构体 |
| GPIO_PinLockConfig | 将 GPIO 引脚配置为引脚锁定功能 |
| GPIO_PinRemapConfig | 将 GPIO 引脚配置为引脚映射功能 |
| GPIO_ReadInputBit | 读取所选输入 GPIO 引脚的状态(1 位) |
| GPIO_ReadInputData | 读取所选输入 GPIO 引脚的状态(1 组，16 位) |
| GPIO_ReadOutputBit | 读取所选输出 GPIO 引脚的状态(1 位) |
| GPIO_ReadOutputData | 读取所选输出 GPIO 引脚的状态(1 组，16 |

| | |
|---|---|
| | 位) |
| GPIO_ResetBits | 复位所选 GPIO 引脚的输出状态 |
| GPIO_SetBits | 置位所选 GPIO 引脚的输出状态 |
| GPIO_WriteBit | 置位或复位所选 GPIO 引脚的输出状态 |

### 1.4 应用实例

【例 1.4.1】编写 GPIO 配置函数，并将 PC0 和 PC2 配置为推拉输出、输出最高速度为 50MHz，将 PE0 和 PE1 配置为同样的配置。实例代码如代码清单 1.4.1 所示。

代码清单 1.4.1 实例代码

```
/**
  * @功能：配置 GPIO 端口
  * @输入参数：无
  * @返回值：无
  */
void GPIO_Configuration(void)
{
    GPIO_InitPara GPIO_InitStructure;
    RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_GPIOC
|RCC_APB2PERIPH_GPIOE,ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_PIN_0 | GPIO_PIN_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_MODE_OUT_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_SPEED_50MHZ;
    GPIO_Init(GPIOC,&GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_PIN_0 | GPIO_PIN_1;
    GPIO_Init(GPIOE,&GPIO_InitStructure);
}
```

## 二、ADC 模块

ADC 模块固件库文件为 gd32f10x_adc.c 和 gd32f10x_adc.h，该 ADC 模块固件库提供了与 ADC 模块相关的配置及功能，包括 ADC 初始化、ADC 使能及禁用等。

### 2.1 ADC 寄存器

ADC 模块寄存器的定义如代码清单 2.1.1 所示。

代码清单 2.1.1 ADC 模块寄存器定义代码清单

```
typedef struct
{
    __IO uint32_t STR;
    __IO uint32_t CTLR1;
    __IO uint32_t CTLR2;
    __IO uint32_t SPT1;
    __IO uint32_t SPT2;
    __IO uint32_t ICOS1;
    __IO uint32_t ICOS2;
    __IO uint32_t ICOS3;
    __IO uint32_t ICOS4;
    __IO uint32_t AWHT;
    __IO uint32_t AWLT;
    __IO uint32_t RSQ1;
    __IO uint32_t RSQ2;
    __IO uint32_t RSQ3;
    __IO uint32_t ISQ;
    __IO uint32_t IDTR1;
    __IO uint32_t IDTR2;
    __IO uint32_t IDTR3;
    __IO uint32_t IDTR4;
    __IO uint32_t RDTR;
} ADC_TypeDef;
```

ADC1 的寄存器映射如代码清单 2.1.2 所示。

代码清单 2.1.2 ADC 模块寄存器地址映射

```
#define ADC1                    ((ADC_TypeDef *) ADC1_BASE)
#define ADC1_BASE               (APB2PERIPH_BASE + 0x2400)
#define APB2PERIPH_BASE         (PERIPH_BASE + 0x10000)
#define PERIPH_BASE             ((uint32_t)0x40000000)
```

## 2.2 ADC 初始化结构体类型 ADC_InitPara

ADC_InitPara 结构体类型定义如代码清单 2.2.1 所示。

代码清单 2.2.1 ADC_InitPara 结构体类型定义代码

```
typedef struct
{
    uint32_t ADC_Trig_External;
    uint8_t ADC_Channel_Number;
    uint32_t ADC_Data_Align;
    TypeState ADC_Mode_Scan;
    uint32_t ADC_Mode;
    TypeState ADC_Mode_Continuous;
}ADC_InitPara;
```

ADC_InitPara 结构体成员变量说明如表 2.2.1 所示。

表 2.2.1 ADC_InitPara 结构体成员变量描述表

| 类型 | 成员变量 | 功能描述 | 举例 |
|---|---|---|---|
| uint32_t | ADC_Trig_External | 规则组触发源选择 | ADC_EXTERNAL_TRIGGER_MODE_NONE |
| uint8_t | ADC_Channel_Number | ADC 转换通道选择 | 0-15 |
| uint32_t | ADC_Data_Align | ADC 采样对其模式选择，左对齐或右对齐 | ADC_DATAALIGN_RIGHT |
| TypeState | ADC_Mode_Scan | 是否使用扫描模式 | ENABLE |
| uint32_t | ADC_Mode | ADC 操作模式选择，可以配置为独立模式或双 ADC 模式 | ADC_MODE_INDEPENDENT |
| TypeState | ADC_Mode_Continuous | 是否使用连续模式 | ENABLE |

其中，ADC_InitPara-> ADC_Trig_External 成员变量可从 ADC_external_trigger 定义常量中选择，ADC_external_trigger 定义常量的代码如代码清单 2.2.2 所示。

代码清单 2.2.2 ADC_external_trigger 定义常量代码清单

```
#define  ADC_EXTERNAL_TRIGGER_MODE_T1_CC1                    ((uint32_t)0x00000000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T1_CC2                    ((uint32_t)0x00020000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T2_CC2                    ((uint32_t)0x00060000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T3_TRGO                   ((uint32_t)0x00080000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T4_CC4                    ((uint32_t)0x000A0000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_EXT_IT11_T8_TRGO          ((uint32_t)0x000C0000)
/*!< Only used in ADC1 and ADC2 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T1_CC3                    ((uint32_t)0x00040000)
/*!< Used in ADC1,ADC2 and ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_NONE                      ((uint32_t)0x000E0000)
/*!< Used in ADC1,ADC2 and ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T3_CC1                    ((uint32_t)0x00000000)
/*!< Only used in ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T2_CC3                    ((uint32_t)0x00020000)
/*!< Only used in ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T8_CC1                    ((uint32_t)0x00060000)
/*!< Only used in ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T8_TRGO                   ((uint32_t)0x00080000)
/*!< Only used in ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T5_CC1                    ((uint32_t)0x000A0000)
/*!< Only used in ADC3 */
#define  ADC_EXTERNAL_TRIGGER_MODE_T5_CC3                    ((uint32_t)0x000C0000)
/*!< Only used in ADC3 */
```

ADC_InitPara-> ADC_Channel_Number 可选择 0~15 的常数，代表所选择 ADC 模块的通道数目。

ADC_InitPara-> ADC_Data_Align 成员变量可从 ADC_data_align 定义常量中选择，ADC_data_align 定义常量的代码如代码清单 2.2.3 所示。

代码清单 2.2.3 ADC_data_align 定义常量的代码

```
#define ADC_DATAALIGN_RIGHT                                  ((uint32_t)0x00000000)
#define ADC_DATAALIGN_LEFT                                   ((uint32_t)0x00000800)
```

ADC_InitPara-> ADC_Mode_Scan 成员变量用于设置是否使用扫描模式，扫描采样模式是在多通道采样中使用，因此多通道采样时，需将该成员变量设置为 ENABLE。

ADC_InitPara-> ADC_Mode 成员变量可从 ADC_mode 定义常量中选择，ADC_mode 定义常量的代码如代码清单 2.2.4 所示。

代码清单 2.2.4 ADC_mode 定义常量的代码

```
#define ADC_MODE_INDEPENDENT                        ((uint32_t)0x00000000)
#define ADC_MODE_REGINSERTSIMULT                    ((uint32_t)0x00010000)
#define ADC_MODE_REGSIMULT_ALTERTRIG                ((uint32_t)0x00020000)
#define ADC_MODE_INSERTSIMULT_FASTINTERL            ((uint32_t)0x00030000)
#define ADC_MODE_INSERTSIMULT_SLOWINTERL            ((uint32_t)0x00040000)
#define ADC_MODE_INSERTSIMULT                       ((uint32_t)0x00050000)
#define ADC_MODE_REGSIMULT                          ((uint32_t)0x00060000)
#define ADC_MODE_FASTINTERL                         ((uint32_t)0x00070000)
#define ADC_MODE_SLOWINTERL                         ((uint32_t)0x00080000)
#define ADC_MODE_ALTERTRIG                          ((uint32_t)0x00090000)
```

ADC_InitPara-> ADC_Mode_Continuous 成员变量表示是否使用连续采样模式，如需使用连续采样模式，需将该成员变量设置为 ENABLE.

## 2.3 ADC 模块库函数

ADC 模块库函数列表如表 2.3.1 所示。

表 2.3.1 ADC 模块库函数列表

| 函数名称 | 功能描述 |
|---|---|
| ADC_AnalogWatchdog_Enable | 使能或禁用模拟看门狗 |
| ADC_AnalogWatchdogSingleChannel_Config | 配置模拟看门狗的采样通道及所需 ADCx 模块 |
| ADC_AnalogWatchdogThresholds_Config | 配置模拟看门狗的高、低电压阈值 |
| ADC_AutoInsertedConv_Enable | 使能或禁用 ADC 注入组自动转换 |
| ADC_Calibration | ADC 校准 |
| ADC_ClearBitState | 清除 ADC 状态标志位(ADC_FLAG_AWE/ ADC_FLAG_EOC/ ADC_FLAG_EOIC/ ADC_FLAG_STIC/ ADC_FLAG_STRC) |
| ADC_ClearIntBitState | 清除 ADC 中断标志位(ADC_INT_EOC/ ADC_INT_AWE/ ADC_INT_EOIC) |
| ADC_DeInit | 复位 ADC 模块，并将 ADC_InitPara 结构体中填入缺省值 |
| ADC_DiscMode_Enable | 使能或禁用间断模式 |
| ADC_DiscModeChannelCount_Config | 配置 ADCx 工作在间断模式下的规则组转换个数 |
| ADC_DMA_Enable | 使能或禁用 ADCx 的 DMA 请求 |
| ADC_Enable | 使能或禁用 ADCx 接口 |
| ADC_ExternalTrigConv_Enable | 使能或禁用 ADCx 的外部触发 |
| ADC_ExternalTrigInsertedConv_Config | 配置 ADCx 注入通道组的外部触发源 |
| ADC_ExternalTrigInsertedConv_Enable | 使能或禁用 ADCx 通过外部触发源进行注入通道组转换 |
| ADC_GetBitState | 检查 ADCx 标志位状态(置位或复位) |
| ADC_GetConversionValue | 获得 ADCx 规则通道组转换数据 |
| ADC_GetDualModeConversionValue | 在双 ADC 模式下，返回 ADC1 和 ADC2 转换数据 |
| ADC_GetInsertedConversionValue | 获得 ADC 注入通道组转换数据 |

| | |
|---|---|
| ADC_GetIntState | 检查指定的 ADC 中断状态(置位或复位) |
| ADC_GetSoftwareStartConvBitState | 得到 ADCx 软件触发转换的状态(置位或复位) |
| ADC_GetSoftwareStartInsertedConvCmdBitState | 得到 ADC 软件触发注入通道组转换状态(置位或复位) |
| ADC_Init | 初始化 ADC 接口模块成员参数 |
| ADC_InsertedChannel_Config | 配置注入组通道及采样时间(总转换时间=采样世间+12.5cycles) |
| ADC_InsertedDiscMode_Enable | 对 ADCx 注入通道组使能或禁用间断模式 |
| ADC_InsertedSequencerLength_Config | 配置注入通道组序列长度 |
| ADC_INTConfig | 使能或禁用 ADC 中断 |
| ADC_RegularChannel_Config | 配置规则组通道及采样周期 |
| ADC_SetInsertedOffset | 设置注入通道组转换数据的偏移量基值 |
| ADC_SoftwareStartConv_Enable | 使能或禁用 ADC 软件触发转换 |
| ADC_SoftwareStartInsertedConv_Enable | 使能或禁用启动 ADC 注入通道转换 |
| ADC_TempSensorVrefint_Enable | 使能或禁用片内温度传感器和内部参考电压通道 |

## 2.4 应用实例

【例 2.4.1】编写 ADC 采样程序，利用 ADC1 进行采样，并利用 DMA1 将采样的数据搬到内存中。

主函数见代码清单 2.4.1.

代码清单 2.4.1

```
#include "gd32f10x.h"
#include <stdlib.h>
#include <stdio.h>
#define ADC1_DR_Address        ((uint32_t)0x4001244C)
__IO uint16_t ADCConvertedValue;
ADC_InitPara ADC_InitStructure;
DMA_InitPara DMA_InitStructure;
void RCC_Configuration(void);
void GPIO_Configuration(void);
int main(void)
{
      RCC_Configuration();
      GPIO_Configuration();
      DMA_DeInit(DMA1_CHANNEL1);
      DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
      DMA_InitStructure.DMA_MemoryBaseAddr = (uint32_t)&ADCConvertedValue;
      DMA_InitStructure.DMA_DIR = DMA_DIR_PERIPHERALSRC;
      DMA_InitStructure.DMA_BufferSize = 1;
      DMA_InitStructure.DMA_PeripheralInc = DMA_PERIPHERALINC_DISABLE;
      DMA_InitStructure.DMA_MemoryInc = DMA_MEMORYINC_DISABLE;
      DMA_InitStructure.DMA_PeripheralDataSize = DMA_PERIPHERALDATASIZE_HALFWORD;
      DMA_InitStructure.DMA_MemoryDataSize = DMA_MEMORYDATASIZE_HALFWORD;
      DMA_InitStructure.DMA_Mode = DMA_MODE_CIRCULAR;
      DMA_InitStructure.DMA_Priority = DMA_PRIORITY_HIGH;
      DMA_InitStructure.DMA_MTOM = DMA_MEMTOMEM_DISABLE;
      DMA_Init(DMA1_CHANNEL1, &DMA_InitStructure);
      DMA_Enable(DMA1_CHANNEL1, ENABLE);
      ADC_InitStructure.ADC_Mode = ADC_MODE_INDEPENDENT;
      ADC_InitStructure.ADC_Mode_Scan = ENABLE;
      ADC_InitStructure.ADC_Mode_Continuous = ENABLE;
      ADC_InitStructure.ADC_Trig_External = ADC_EXTERNAL_TRIGGER_MODE_NONE;
      ADC_InitStructure.ADC_Data_Align = ADC_DATAALIGN_RIGHT;
      ADC_InitStructure.ADC_Channel_Number = 1;
      ADC_Init(ADC1,&ADC_InitStructure);
      ADC_RegularChannel_Config(ADC1, ADC_CHANNEL_13, 1, ADC_SAMPLETIME_55POINT5);
      ADC_ExternalTrigConv_Enable(ADC1,ENABLE);
      ADC_DMA_Enable(ADC1,ENABLE);
      ADC_Enable(ADC1,ENABLE);
      ADC_Calibration(ADC1);
      ADC_SoftwareStartConv_Enable(ADC1,ENABLE);
      while (1)
      {
      }
}
```

系统时钟配置函数见代码清单 2.4.2。

代码清单 2.4.2

```
void RCC_Configuration(void)
{
    /* ADCCLK = PCLK2/6 */
    RCC_ADCCLKConfig(RCC_ADCCLK_APB2_DIV6);
    /* Enable DMA1 and GPIOC clock */
    RCC_AHBPeriphClock_Enable(RCC_AHBPERIPH_DMA1, ENABLE);
    RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_GPIOC, ENABLE);
    /* Enable ADC1 clock */
    RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_ADC1, ENABLE);
}
```

GPIO 配置函数见代码清单 2.4.3.

代码清单 2.4.3

```
void GPIO_Configuration(void)
{
    GPIO_InitPara GPIO_InitStructure;
    /* Configure PC3 (ADC Channel13) as analog input -------------------------*/
    GPIO_InitStructure.GPIO_Pin = GPIO_PIN_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_MODE_AIN;
    GPIO_InitStructure.GPIO_Speed = GPIO_SPEED_50MHZ;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

# 三、EXTI 模块

EXTI 模块固件库文件为 gd32f10x_exti.c 和 gd32f10x_exti.h，该固件库文件提供与 EXTI 模块相关的配置函数，包括

## 3.1 EXTI 寄存器

EXTI 寄存器的定义如代码清单 3.1.1 所示。

代码清单 3.1.1

```
typedef struct
{
    __IO uint32_t IER;                    /*!<EXTI Interrupt enable register, Address offset: 0x00 */
    __IO uint32_t EER;                    /*!<EXTI Event enable register, Address offset: 0x04 */
    __IO uint32_t RTE;                    /*!<EXTI Rising edge trigger enable register, Address offset: 0x08 */
    __IO uint32_t FTE;                    /*!<EXTI Falling edge trigger enable register, Address offset: 0x0C */
    __IO uint32_t SIE;                    /*!<EXTI Software interrupt event register, Address offset: 0x10 */
    __IO uint32_t PD;                     /*!<EXTI Pending register, Address offset: 0x14 */
} EXTI_TypeDef;
```

EXTI 模块寄存器映射如代码清单 3.1.2 所示。

代码清单 3.1.2

```
#define EXTI                ((EXTI_TypeDef *) EXTI_BASE)
#define EXTI_BASE           (APB2PERIPH_BASE + 0x0400)
#define APB2PERIPH_BASE     (PERIPH_BASE + 0x10000)
#define PERIPH_BASE         ((uint32_t)0x40000000)
```

## 3.2 EXTI 模块初始化结构体类型 EXTI_InitPara

EXTI 模块初始化结构体类型 EXTI_InitPara 的定义如代码清单 3.2.1 所示。

代码清单 3.2.1

```
typedef struct
{
    uint32_t EXTI_LINE;
    EXTI_ModePara EXTI_Mode;
    EXTI_TriggerPara EXTI_Trigger;
    TypeState EXTI_LINEEnable;
}EXTI_InitPara;
```

EXTI_InitPara 结构体类型的成员变量有三个，具体说明见表 3.2.1 所示。

表 3.2.1 EXTI_InitPara 结构体成员变量说明

| 类型 | 成员变量 | 功能描述 | 举例 |
|---|---|---|---|
| uint32_t | EXTI_LINE | EXTI 外部中断通道选择 | EXTI_LINE0 |
| EXTI_ModePara | EXTI_Mode | 外部中断通道工作模式(中断或事件) | EXTI_Mode_Interrupt |
| EXTI_TriggerPara | EXTI_Trigger | 边沿触发设置 | EXTI_Trigger_Rising ENABLE |

| | | | |
|---|---|---|---|
| TypeState | EXTI_LINEEnable | 使能或禁用外部中断通道 | ENABLE |

EXTI_InitPara-> EXTI_LINE 成员可从 EXTI_lines 所定义的宏定义中获得，EXTI_lines 定义的宏定义如代码清单 3.2.2 所示，其代表可以选择的外部中断通道，从 LINE0~LINE19，前 16 个连接 GPIOx_pin0~GPIOx_pin15，EXTI16 连接 PVD(电源电压检测)输出，EXTI17 连接 RTC 闹钟事件，EXT18 连接 USB 唤醒事件。

代码清单 3.2.2 EXTI_lines 宏定义代码

```
/** @defgroup EXTI_lines
  * @{
  */
#define EXTI_LINE0          ((uint32_t)0x00000001)          /*!< External interrupt line 0 */
#define EXTI_LINE1          ((uint32_t)0x00000002)          /*!< External interrupt line 1 */
#define EXTI_LINE2          ((uint32_t)0x00000004)          /*!< External interrupt line 2 */
#define EXTI_LINE3          ((uint32_t)0x00000008)          /*!< External interrupt line 3 */
#define EXTI_LINE4          ((uint32_t)0x00000010)          /*!< External interrupt line 4 */
#define EXTI_LINE5          ((uint32_t)0x00000020)          /*!< External interrupt line 5 */
#define EXTI_LINE6          ((uint32_t)0x00000040)          /*!< External interrupt line 6 */
#define EXTI_LINE7          ((uint32_t)0x00000080)          /*!< External interrupt line 7 */
#define EXTI_LINE8          ((uint32_t)0x00000100)          /*!< External interrupt line 8 */
#define EXTI_LINE9          ((uint32_t)0x00000200)          /*!< External interrupt line 9 */
#define EXTI_LINE10         ((uint32_t)0x00000400)          /*!< External interrupt line 10 */
#define EXTI_LINE11         ((uint32_t)0x00000800)          /*!< External interrupt line 11 */
#define EXTI_LINE12         ((uint32_t)0x00001000)          /*!< External interrupt line 12 */
#define EXTI_LINE13         ((uint32_t)0x00002000)          /*!< External interrupt line 13 */
#define EXTI_LINE14         ((uint32_t)0x00004000)          /*!< External interrupt line 14 */
#define EXTI_LINE15         ((uint32_t)0x00008000)          /*!< External interrupt line 15 */
#define EXTI_LINE16         ((uint32_t)0x00010000)          /*!< External interrupt line 16
                                                            Connected to the LVD */
#define EXTI_LINE17         ((uint32_t)0x00020000)          /*!< External interrupt line 17
                                                              Connected to the RTC Alarm */
#define EXTI_LINE18         ((uint32_t)0x00040000)          /*!< External interrupt line 18
                                                            Connected to the USB Wakeup */
#define EXTI_LINE19         ((uint32_t)0x00080000)          /*!< External interrupt line 19
                                                            Connected to the Ethernet Wakeup */
```

EXTI_InitPara-> EXTI_Mode 成员变量可从 EXTI_ModePara 共用体中获得，EXTI_ModePara 共用体程序代码如代码清单 3.2.3 所示。其中，EXTI_Mode_Interrupt 代表选择中断模式，EXTI_Mode_Event 代表选择事件模式。

代码清单 3.2.3 EXTI_ModePara 共用体程序代码

```
typedef enum
{
    EXTI_Mode_Interrupt = 0x00,
    EXTI_Mode_Event = 0x04
}EXTI_ModePara;
```

EXTI_InitPara-> EXTI_Trigger 成员变量可从共用体 EXTI_TriggerPara 中获得，EXTI_TriggerPara 共用体程序代码如代码清单 3.2.4 所示，其中，EXTI_Trigger_Rising 代表上升沿触发外部中断，EXTI_Trigger_Falling 表示下降沿触发中断，EXTI_Trigger_Rising_Falling 表示上升沿和下降沿均触发中断。

代码清单 3.2.4 EXTI_TriggerPara 共用体程序代码

```
typedef enum
{
    EXTI_Trigger_Rising = 0x08,
    EXTI_Trigger_Falling = 0x0C,
    EXTI_Trigger_Rising_Falling = 0x10
}EXTI_TriggerPara;
```

EXTI_InitPara->EXTI_LINEEnable 成员变量表示中断通道使能或禁用，可以设置为 ENABLE 或 DISABLE。

## 3.3 EXTI 模块库函数

EXTI 模块库函数说明如表 3.3.1 所示。

表 3.3.1 EXTI 模块库函数说明

| 函数名称 | 功能描述 |
|---|---|
| EXTI_ClearBitState | 清除所选 EXTI 通道状态标志位 |
| EXTI_ClearIntBitState | 清除所选 EXTI 通道中断标志位 |
| EXTI_DeInit | 复位 EXTI 外设并初始化 EXTI_InitPara 结构体 |
| EXTI_GetBitState | 获得所选 EXTI 通道状态标志位 |
| EXTI_GetIntBitState | 获得所选 EXTI 通道中断标志位 |
| EXTI_Init | 初始化 EXTI 模块寄存器 |
| EXTI_SWINT_Enable | 使能所选 EXTI 模块的软件中断或事件请求 |

## 3.4 应用实例

【例 3.4.1】编程实现利用外部中断反转 LED。

本实例主程序代码如代码清单 3.4.1 所示。

代码清单 3.4.1

```
#include "gd32f10x.h"
/* Private variables ---------------------------------------------------------*/
GPIO_InitPara GPIO_InitStructure;
EXTI_InitPara EXTI_InitStructure;
NVIC_InitPara NVIC_InitStructure;
/* Private function prototypes -----------------------------------------------*/
void EXTI0_Config(void);
void EXTI14_Config(void);
int main(void)
{
    /* Enable GPIOC clock */
    RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_GPIOC, ENABLE);
    /* Configure the LED2 and LED3 GPIO */
    GPIO_InitStructure.GPIO_Pin = GPIO_PIN_0 | GPIO_PIN_2;
    GPIO_InitStructure.GPIO_Mode = GPIO_MODE_OUT_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_SPEED_50MHZ;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    /* Turn on LED2 and LED3 */
    GPIO_SetBits(GPIOC, GPIO_PIN_0);
    GPIO_SetBits(GPIOC, GPIO_PIN_2);
    /* Configure the EXTI line0 and EXTI line14 */
    EXTI_DeInit(&EXTI_InitStructure);
    EXTI0_Config();
    EXTI14_Config();
    EXTI_SWINT_Enable(EXTI_LINE0);
    while(1)
    {
    }
}
```

EXTI0 通道配置函数如代码清单 3.4.2 所示。

代码清单 3.4.2

```
void EXTI0_Config(void)
{
    /* Enable GPIOC and AFIO clock */
    RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_GPIOA |
                                    RCC_APB2PERIPH_AF, ENABLE);
    /* Configure PA0 pin */
    GPIO_InitStructure.GPIO_Pin = GPIO_PIN_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_MODE_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /* Connect EXTI Line0 to PA0 pin */
    GPIO_EXTILineConfig(GPIO_PORT_SOURCE_GPIOA, GPIO_PINSOURCE0);
    /* Configure EXTI Line0 and its interrupt to the lowest priority*/
    EXTI_InitStructure.EXTI_LINE = EXTI_LINE0;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LINEEnable = ENABLE;
    EXTI_Init(&EXTI_InitStructure);
    NVIC_InitStructure.NVIC_IRQ = EXTI0_IRQn;
    NVIC_InitStructure.NVIC_IRQPreemptPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQSubPriority = 0x0F;
    NVIC_InitStructure.NVIC_IRQEnable = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}
```

EXTI14 通道配置函数如代码清单 3.4.3 所示。

代码清单 3.4.3

```
void EXTI14_Config(void)
{
      /* Enable GPIOB and AFIO clock */
      RCC_APB2PeriphClock_Enable(RCC_APB2PERIPH_GPIOB |
                                          RCC_APB2PERIPH_AF, ENABLE);
    /* Configure PB14 pin */
      GPIO_InitStructure.GPIO_Pin = GPIO_PIN_14;
      GPIO_InitStructure.GPIO_Mode = GPIO_MODE_IN_FLOATING;
      GPIO_Init(GPIOB, &GPIO_InitStructure);
      /* Connect EXTI Line14 to PB14 pin */
      GPIO_EXTILineConfig(GPIO_PORT_SOURCE_GPIOB, GPIO_PINSOURCE14);
      /* Configure EXTI Line14 and its interrupt to the lowest priority*/
      EXTI_InitStructure.EXTI_LINE = EXTI_LINE14;
      EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
      EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
      EXTI_InitStructure.EXTI_LINEEnable = ENABLE;
      EXTI_Init(&EXTI_InitStructure);
      NVIC_InitStructure.NVIC_IRQ = EXTI15_10_IRQn;
      NVIC_InitStructure.NVIC_IRQPreemptPriority = 0x0F;
      NVIC_InitStructure.NVIC_IRQSubPriority = 0x0F;
      NVIC_InitStructure.NVIC_IRQEnable = ENABLE;
      NVIC_Init(&NVIC_InitStructure);
}
```

EXTI0 以及 EXTI14 的中断服务函数如代码清单 3.4.4 所示，在中断中实现对 GPIO 引脚的控制。

代码清单 3.4.4

```
void EXTI0_IRQHandler(void)
{
    if(EXTI_GetIntBitState(EXTI_LINE0) != RESET)
    {
        GPIO_WriteBit(GPIOC, GPIO_PIN_0,
                      (BitState)((1-GPIO_ReadOutputBit(GPIOC, GPIO_PIN_0))));
        EXTI_ClearIntBitState(EXTI_LINE0);
    }
}
void EXTI15_10_IRQHandler(void)
{
    if(EXTI_GetIntBitState(EXTI_LINE14) != RESET)
    {
        GPIO_WriteBit(GPIOC, GPIO_PIN_2,
                      (BitState)((1-GPIO_ReadOutputBit(GPIOC, GPIO_PIN_2))));
        EXTI_ClearIntBitState(EXTI_LINE14);
    }
}
```