重庆理工大学电子信息与自动化学院

HT32 库函数基础

盛群杯专题讲座

万文略

2013-4-1

目录

利用 HT32 库函数进行 HT32F176x MCU 应用程序设计有很多优点，不仅能从繁杂的寄存器设置中解脱出来从而进行快速的程序设计，而且通过阅读专业人士编写的库函数源代码，能够学到优秀的编程风格，提高自己的编程水平。提供库函数支持是目前各 MCU 公司推广自己的 MCU 的重要手段，学习了一个公司的库函数后，再学习其他公司的 MCU 库函数就容易很多了。

使用库函数编程为开发人员进行程序设计提供了很多便利，但是使用库函数编程的代码效率比直接使用寄存器进行程序设计的代码效率稍低，在有更高的实时性要求的程序中可以使用直接操作寄存器的方式编程。

学习库函数编程从 MCU 的头文件开始，对于 HT32F176xMCU，学习库函数编程要从 ht32f175x_275x.h 文件开始。

这个文件定义了库函数使用的数据类型，外设及村级结构数据类型及库函数使用的符号常量。

1、 库函数使用的基本数据类型

1.1、 缩写词

ADC 模数转换器模块（ADC driver modules. ）
BFTM 基本定时器模块（BFTM driver modules.）
CKCU 时钟模块（CKCU driver modules. ）
CMP_OP 运算放大器/比较器模块（CMP_OP driver modules. ）
EXTI 外部中断模块（EXTI driver modules. ）
FLASH FLASH 模块（FLASH driver modules. ）
GPIO 通用输入输出口模块（GPIO driver modules. ）
GPTM 通用定时器模块（GPTM driver modules.）
I2C I2C 总线模块（I2C driver modules. ）
MCTM 马达控制定时器模块（MCTM driver modules. ）
MISC 杂项模块（MISC driver modules. ）
PDMA 外设直接存储器存取模块（PDMA driver modules. ）
PWRCU 电源管理模块（PWRCU driver modules. ）
RSTCU 复位模块 （RSTCU driver modules. ）
RTC 实时时钟模块（RTC driver modules. ）
SCI 智能卡接口模块（SCI driver modules. ）
SPI SPI 总线模块（SPI driver modules. ）
USART 串口模块 （USART driver modules. ）
WDT 看门狗模块（WDT driver modules. ）
CSIF CMOS 图像传感器接口模块（ CSIF driver modules.）

1.2、 变量的数据类型

```
typedef enum IRQn    IRQn_Type
typedef signed long    s32
typedef signed short    s16
typedef signed char    s8
typedef const int32_t    sc32
typedef const int16_t    sc16
typedef const int8_t    sc8
typedef __IO int32_t    vs32
typedef __IO int16_t    vs16
typedef __IO int8_t    vs8
typedef __I int32_t    vsc32
typedef __I int16_t    vsc16
typedef __I int8_t    vsc8
typedef unsigned long    u32
typedef unsigned short    u16
typedef unsigned char    u8
typedef unsigned long const    uc32
typedef unsigned short const    uc16
typedef unsigned char const    uc8
typedef __IO uint32_t    vu32
typedef __IO uint16_t    vu16
typedef __IO uint8_t    vu8
typedef __I uint32_t    vuc32
typedef __I uint16_t    vuc16
typedef __I uint8_t    vuc8
typedef enum EventStatus    ControlStatus
```

1.3、 枚举

```
enum      IRQn {
   NonMaskableInt_IRQn = -14, HardFault_IRQn = -13,
emoryManagement_IRQn = -12, BusFault_IRQn = -11,
   UsageFault_IRQn = -10, SVCall_IRQn = -5, DebugMonitor_IRQn = -4,
   PendSV_IRQn = -2,
   SysTick_IRQn = -1, CKRDY_IRQn = 0, LVD_IRQn = 1, BOD_IRQn = 2,
   WDT_IRQn = 3, RTC_IRQn = 4, FLASH_IRQn = 5, EVWUP_IRQn = 6,
   LPWUP_IRQn = 7, EXTI0_IRQn = 8, EXTI1_IRQn = 9, EXTI2_IRQn = 10,
   EXTI3_IRQn = 11, EXTI4_IRQn = 12, EXTI5_IRQn = 13, EXTI6_IRQn = 14,
   EXTI7_IRQn = 15, EXTI8_IRQn = 16, EXTI9_IRQn = 17, EXTI10_IRQn = 18,
   EXTI11_IRQn = 19, EXTI12_IRQn = 20, EXTI13_IRQn = 21, EXTI14_IRQn = 22,
   EXTI15_IRQn = 23, COMP_IRQn = 24, ADC_IRQn = 25, MCTMBRK_IRQn = 27,
   MCTMUP_IRQn = 28, MCTMTR_IRQn = 29, MCTMCC_IRQn = 30,
   GPTM0_IRQn = 35,
```

```
  GPTM1_IRQn = 36, BFTM0_IRQn = 41, BFTM1_IRQn = 42, I2C0_IRQn = 43,
  I2C1_IRQn = 44, SPI0_IRQn = 45, SPI1_IRQn = 46, USART0_IRQn = 47,
  USART1_IRQn = 48, SCI_IRQn = 51, USB_IRQn = 53, PDMACH0_IRQn = 55,
  PDMACH1_IRQn = 56, PDMACH2_IRQn = 57, PDMACH3_IRQn = 58,
  PDMACH4_IRQn = 59,
  PDMACH5_IRQn = 60, PDMACH6_IRQn = 61, PDMACH7_IRQn = 62,
  PDMACH8_IRQn = 63,
  PDMACH9_IRQn = 64, PDMACH10_IRQn = 65, PDMACH11_IRQn = 66,
  CSIF_IRQn = 67
}
enum     EventStatus { DISABLE = 0, ENABLE = !DISABLE }
enum     bool { FALSE = 0, TRUE = !FALSE }
enum     FlagStatus { RESET = 0, SET = !RESET }
enum     ErrStatus { ERROR = 0, SUCCESS = !ERROR }
```

使用 IRQn 的枚举后，所有 HT32 的中断源都可以用预定义的符号表示，不用再查阅手册。

即为编程提供方便，又增加了程序的可读性。

使用 EventStatus，bool ，FlagStatus，ErrStatus 后可以用符号表示事件状态，逻辑状态，标志状态，错误状态。这些状态即可以是库函数的调用参数，也可以是库函数的返回值。

1.4、 外设模块控制寄存器数据结构类型

外设模块控制寄存器数据结构类型

库函数中用结构体（struct）定义了全部外设的寄存器体，因为篇幅的关系这里只给出其中的部分。

```
typedef struct
{
      /* USART0: 0x40000000     */
      /* USART1: 0x40040000    */
  union {
  __IO uint32_t RBR;     /*!< 0x000          Receive Buffer Register    */
  __IO uint32_t TBR;     /*!< 0x000          Transmit Holding Register */
   };
  __IO uint32_t IER;         /*!< 0x004        Interrupt Enable Register */
  __IO uint32_t IIR;       /*!< 0x008        Interrupt Identification Register/FIFO
  Control Register */
   __IO uint32_t FCR;        /*!< 0x00C       FIFO Control Register */
   __IO uint32_t LCR;        /*!< 0x010       Line Control Register*/
   __IO uint32_t MCR;         /*!< 0x014       Modem Control Register    */
   __IO uint32_t LSR;        /*!< 0x018       Line Status Register*/
   __IO uint32_t MSR;         /*!< 0x01C       Modem Status Register*/
   __IO uint32_t TPR;        /*!< 0x020       Timing Parameter Register*/
```

```
    __IO uint32_t MDR;       /*!< 0x024       Mode Register */
    __IO uint32_t ICR;       /*!< 0x028       IrDA Register*/
    __IO uint32_t RCR;       /*!< 0x02C       RS485 Control Register*/
    __IO uint32_t SCR;       /*!< 0x030       Synchronous Control Register*/
    __IO uint32_t FSR;       /*!< 0x034       FIFO Status Register*/
    __IO uint32_t DLR;       /*!< 0x038       Divisor Latch Register*/
    __IO uint32_t DTR;       /*!< 0x040       Debug/Test Register*/
 } USART_TypeDef;
```

通过这个数据类型定义可以看出

0x40000000 是 USART0 寄存器的基地址，也就是说从这个地址开始的 N 个单元连续分布这 USART0 的寄存器。0x40040000 是 USART1 的基地址。

```
union {
   __IO uint32_t RBR;     /*!< 0x000          Receive Buffer Register    */
   __IO uint32_t TBR;     /*!< 0x000          Transmit Holding Register */
    };
```

这个联合体表示接收寄存器（RBR）和发送寄存器（TBR）2 个寄存器但是使用同一个物理地址。

/*!< 0x004 Interrupt Enable Register */注释中的数字 0x004 表示的是 IER 寄存器距离基地址的偏移地址。类推，其他的本文不再说明。

对于其他外设模块也类似地定义了外设寄存器数据结构类型，格式是 XXX_TypeDef

其中 XXX 是外设名称的缩写

1.5、 外设寄存器声明

```
#define USART0                   ((USART_TypeDef *) USART0_BASE)
#define USART1                   ((USART_TypeDef *) USART1_BASE)
#define SPI0                    ((SPI_TypeDef *) SPI0_BASE)
#define SPI1                    ((SPI_TypeDef *) SPI1_BASE)
#define ADC                      ((ADC_TypeDef *) ADC_BASE)
#define CMP_OP0                   ((CMP_OP_TypeDef *) CMP_OP0_BASE)
#define CMP_OP1                   ((CMP_OP_TypeDef *) CMP_OP1_BASE)
#define AFIO                    ((AFIO_TypeDef *) AFIO_BASE)
#define GPIOA                    ((GPIO_TypeDef *) GPIOA_BASE)
#define GPIOB                    ((GPIO_TypeDef *) GPIOB_BASE)
#define GPIOC                    ((GPIO_TypeDef *) GPIOC_BASE)
#define GPIOD                    ((GPIO_TypeDef *) GPIOD_BASE)
#define GPIOE                    ((GPIO_TypeDef *) GPIOE_BASE)
#define EXTI                    ((EXTI_TypeDef *) EXTI_BASE)
#define MCTM                      ((MCTM_TypeDef *) MCTM_BASE)
#define SCI                     ((SCI_TypeDef *) SCI_BASE)
#define I2C0                    ((I2C_TypeDef *) I2C0_BASE)
```

```
#define I2C1                    ((I2C_TypeDef *) I2C1_BASE)
#define USB                      ((USB_TypeDef *) USB_BASE)
#define USBEP0                   ((USBEP_TypeDef *) USB_EP0_BASE)
#define USBEP1                   ((USBEP_TypeDef *) USB_EP1_BASE)
#define USBEP2                   ((USBEP_TypeDef *) USB_EP2_BASE)
#define USBEP3                   ((USBEP_TypeDef *) USB_EP3_BASE)
#define USBEP4                   ((USBEP_TypeDef *) USB_EP4_BASE)
#define USBEP5                   ((USBEP_TypeDef *) USB_EP5_BASE)
#define USBEP6                   ((USBEP_TypeDef *) USB_EP6_BASE)
#define USBEP7                   ((USBEP_TypeDef *) USB_EP7_BASE)
#define WDT                      ((WDT_TypeDef *) WDT_BASE)
#define RTC                     ((RTC_TypeDef *) RTC_BASE)
#define PWRCU                    ((PWRCU_TypeDef *) PWRCU_BASE)
#define GPTM0                    ((GPTM_TypeDef *) GPTM0_BASE)
#define GPTM1                    ((GPTM_TypeDef *) GPTM1_BASE)
#define BFTM0                    ((BFTM_TypeDef *) BFTM0_BASE)
#define BFTM1                    ((BFTM_TypeDef *) BFTM1_BASE)
#define FLASH                   ((FLASH_TypeDef *) FLASH_BASE)
#define CKCU                     ((CKCU_TypeDef *) CKCU_BASE)
#define RSTCU                    ((RSTCU_TypeDef *) RSTCU_BASE)
#define PDMA                     ((PDMA_TypeDef *) PDMA_BASE)
#define CSIF                    ((CSIF_TypeDef *) CSIF_BASE)
```

外设寄存器数据结构类型应用举例

以 USART0 为例，有了头文件的定义后，程序中访问 USART0 外设的 RBR 寄存器时方法是：

```
USART0->RBR=0x56;
```

2、 CKCU 库函数

CKCU 库函数支持 HT32 的 CKCU 模块的程序操作，包括时钟选择，外设时钟使能，读取时钟状态参数等函数。

2.1、 时钟函数一览

表格 1、时钟库函数一览

void CKCU_DeInit (void)
void CKCU_HSECmd (EventStatus Cmd)
void CKCU_HSICmd (EventStatus Cmd)
void CKCU_PLLCmd (EventStatus Cmd)
void CKCU_PLLInit (CKCU_PLLInitTypeDef *PLLInit)
ErrStatus CKCU_WaitHSEReady (void)

FlagStatus CKCU_GetIntStatus (u8 CKCU_INT)
FlagStatus CKCU_GetClockReadyStatus (u32 CKCU_FLAG)
ErrStatus CKCU_SysClockConfig (CKCU_SW_TypeDef CLKSRC)
u8 CKCU_GetSysClockSource (void)
void CKCU_SetHCLKPrescaler (CKCU_SYSCLKDIV_TypeDef DIV)
void CKCU_SetUSARTPrescaler (CKCU_URPRE_TypeDef URPRE)
void CKCU_SetUSBPrescaler (CKCU_USBPRE_TypeDef USBPRE)
void CKCU_SetADCPrescaler (CKCU_PCLKDIV_TypeDef PCLK_DIVX)
void CKCU_WDTSourceConfig (CKCU_WDTSRC_TypeDef SRC)
void CKCU_GetClocksFrequency (CKCU_ClocksTypeDef *CKCU_Clk)
void CKCU_APBPerip0ClockConfig (u32 CKCU_APBP, ControlStatus Cmd)
void CKCU_APBPerip1ClockConfig (u32 CKCU_APBP, ControlStatus Cmd)
void CKCU_CKMCmd (ControlStatus Cmd)
void CKCU_PSRCWKUPCmd (ControlStatus Cmd)
void CKCU_CKOUTConfig (CKCU_CKOUTInitTypeDef *CKOUTInit)
void CKCU_ClearIntFlag (u8 CKCU_INT)
void CKCU_IntConfig (u32 CKCU_INT, ControlStatus Cmd)
void CKCU_SleepClockConfig (u32 CKCU_CLK, ControlStatus Cmd)
bool CKCU_IS_PLL_USED (CKCU_PLLST_TypeDef Target)
bool CKCU_IS_HSI_USED (CKCU_HSIST_TypeDef Target)
bool CKCU_IS_HSE_USED (CKCU_HSEST_TypeDef Target)
void CKCU_MCUDBGConfig (u16 CKCU_DBGx, ControlStatus Cmd)
void CKCU_BKISOCmd (EventStatus Cmd)
void CKCU_AHBPeripClockConfig (u32 CKCU_CLK, ControlStatus Cmd)

2.2、 库函数说明

2.2.1、 CKCU_APBPeripClockConfig 函数

```
void CKCU_AHBPeripClockConfig   ( u32   CKCU_CLK,
  ControlStatus   Cmd
 )
```

使能或禁止 AHB 外设时钟
参数：
CKCU_CLK,: 指定那个外设时钟被禁止或使能. 这个参数可以是:
CKCU_AHBEN_PDMA
CKCU_AHBEN_CSIF
CKCU_AHBEN_CSIFM
Cmd,: 指定的时钟的新状态. 这个参数可以是:
ENABLE
DISABLE

2.2.2、 CKCU_APBPerip0ClockConfig 函数

```
void CKCU_APBPerip0ClockConfig  ( u32  CKCU_APBP,
  ControlStatus  Cmd
 )
```

Enable or Disable the APB peripheral 0 clock.
使能或禁止 APB 外设 0 时钟
注：APB 外设 0 包括：I2C0（I2C 总线接口 0），I2C1（I2C 总线接口 1），SPI0（SPI 总线 0），SPI1（SPI 总线 1），USART0（串口 0），USART1（串口 1），AFIO（复用 IO），EXIT（外部中断），PA（GPIOA 口），PB（GPIOB 口），PC（GPIOC 口），PD（GPIOD 口），PE（GPIOE 口），SCI（智能卡接口）
参数：
CKCU_APBP,: 指定的 APB 0 外设，这个参数可以是:
CKCU_APBEN0_I2C0, CKCU_APBEN0_I2C1, CKCU_APBEN0_SPI0, CKCU_APBEN0_SPI1, CKCU_APBEN0_USART0, CKCU_APBEN0_USART1, CKCU_APBEN0_AFIO, CKCU_APBEN0_EXTI, CKCU_APBEN0_PA, CKCU_APBEN0_PB, CKCU_APBEN0_PC, CKCU_APBEN0_PD, CKCU_APBEN0_PE, CKCU_APBEN0_SCI
Cmd,: 外设时钟状态，这个参数可以是:
ENABLE
DISABLE
注：使用外设前必须打开外设时钟。

2.2.3、 CKCU_APBPerip1ClockConfig 函数

```
void CKCU_APBPerip1ClockConfig  ( u32  CKCU_APBP,
  ControlStatus  Cmd
 )
```

Enable or Disable the APB peripheral 1 clock.
使能或禁止 APB 1 外设时钟
APB 1 外设包括：MCTM（马达控制定时器），WDT（看门狗定时器），RTC（实时时钟），GPTM0（通用定时器 0），GPTM1（通用定时器 1），USB，BFTM0（基本功能定时器 0），BFTM1（基本功能定时器 1），OPA0（运算放大器/比较器 0），OPA1（运算放大器/比较器 1），ADC（模拟数字转换器）
参数
CKCU_APBP,: 指定的 APB 外设 1. 这个参数可以是:
CKCU_APBEN1_MCTM, CKCU_APBEN1_WDT, CKCU_APBEN1_RTC, CKCU_APBEN1_GPTM0, CKCU_APBEN1_GPTM1, CKCU_APBEN1_USB, CKCU_APBEN1_BFTM0, CKCU_APBEN1_BFTM1, CKCU_APBEN1_OPA0, CKCU_APBEN1_OPA1, CKCU_APBEN1_ADC
Cmd,: 外设时钟状态. 这个参数可以是:

ENABLE

DISABLE

初学时只要掌握上面的三个函数即可，下面的函数暂时不译。

3、 GPIO 库函数

3.1、 GPIO 库函数一览

表格 2

void GPIO_OpenDrainConfig (GPIO_TypeDef *GPIOx, u16 GPIO_PIN, ControlStatus Cmd) **Enable or Disable the open drain function of specified GPIO pins.** **使能或禁止指定 GPIO 引脚的开漏输出**
FlagStatus GPIO_ReadInBit (GPIO_TypeDef *GPIOx, u16 GPIO_PIN) **Get the input data of specified port pin.** **读入指定 GPIO 引脚的状态**
u16 GPIO_ReadInData (GPIO_TypeDef *GPIOx) **Get the input data of specified GPIO port.** **读入指定 GPIO 口的输入状态**
FlagStatus GPIO_ReadOutBit (GPIO_TypeDef *GPIOx, u16 GPIO_PIN) **Get the output data of specified port pin.** **读入指定 GPIO 引脚的输出状态**
u16 GPIO_ReadOutData (GPIO_TypeDef *GPIOx) **Get the output data of specified GPIO port.** **读入指定 GPIO 口的输出状态**
void GPIO_SetOutBits (GPIO_TypeDef *GPIOx, u16 GPIO_PIN) **Set the selected port bits of output data.** **指定的 GPIO 引脚置 1**
void GPIO_ClearOutBits (GPIO_TypeDef *GPIOx, u16 GPIO_PIN) **Clear the selected port bits of output data.** **指定的 GPIO 引脚清零**
void GPIO_WriteOutBits (GPIO_TypeDef *GPIOx, u16 GPIO_PIN, FlagStatus Status) **Set or Clear the selected port bits of data.**
void GPIO_WriteOutData (GPIO_TypeDef *GPIOx, u16 Data) **Put data to the specified GPIO data port.** **指定 GPIO 口输出**
void AFIO_EXTISourceConfig (AFIO_EXTI_CH_Enum Channel, AFIO_ESS_Enum Source) **Select the GPIO pin to be used as EXTI channel.** **选择指定的 GPIO 引脚为中断输入引脚**
void GPIO_PinLock (GPIO_TypeDef *GPIOx, u16 GPIO_PIN) **Lock configuration of GPIO pins.**

	锁定指定引脚的配置
bool GPIO_IsPortLocked (GPIO_TypeDef *GPIOx)	
	Get the lock state of specified GPIO port. 读取指定的引脚锁定状态
bool GPIO_IsPinLocked (GPIO_TypeDef *GPIOx, u16 GPIO_PIN)	
	Get the lock state of specified GPIO port pin. 引脚状态是否锁定
void	AFIO_DeInit (void) Deinitialize the AFIO peripheral registers to their default reset values. 恢复 AFIO 的设置为上电复位时的值
void	AFIO_GPAConfig (u32 AFIO_PIN, u32 AFIO_MODE) Configure alternated mode of GPIOA with specified pins. GPIOA 口的引脚复用设置
void	AFIO_GPBConfig (u32 AFIO_PIN, u32 AFIO_MODE) Configure alternated mode of GPIOB with specified pins. GPIOB 口的引脚复用设置
void	AFIO_GPCConfig (u32 AFIO_PIN, u32 AFIO_MODE) Configure alternated mode of GPIOC with specified pins. GPIOC 口的引脚复用设置
void	AFIO_GPDConfig (u32 AFIO_PIN, u32 AFIO_MODE) Configure alternated mode of GPIOD with specified pins. GPIOD 口的引脚复用设置
void	AFIO_GPEConfig (u32 AFIO_PIN, u32 AFIO_MODE) Configure alternated mode of GPIOE with specified pins. GPIOE 口的引脚复用设置

3.2、 库函数说明

3.2.1、AFIO_DeInit

```
void AFIO_DeInit    ( void      )
```

Deinitialize the AFIO peripheral registers to their default reset values.

函数功能：恢复 AFIO 的设置为上电复位时的值

Return values:

函数返回值

None

无

3.2.2、AFIO_EXTISourceConfig

```
void AFIO_EXTISourceConfig    ( AFIO_EXTI_CH_Enum    Channel,
                                AFIO_ESS_Enum    Source
 )
```

Select the GPIO pin to be used as EXTI channel.
选择 GPIO 引脚用作外部中断输入引脚
Parameters:
参数：
Channel,: Specify the EXTI channel to be configured. This parameter can be AFIO_EXTI_CH_x where x can be 0 ~ 15.
Channel：指定 EXTI 通道配置，这个参数可以是 AFIO_EXTI_CH_x,x 可以是 0~15.
需要注意的是这里的 0~15 代表一个 GPIO 口的端口号码。
Source,: Specify the GPIO port to be used for EXTI channel. This parameter can be AFIO_ESS_Px where x can be A ~ E.
Source,:指定作为外部中断用的 GPIO 口，这个参数可以是 AFIO_ESS_Px,其中 x 可以是 A~E。
Return values:
函数返回值
None
无
举例：如果选择 GPIOB 口的 PB6 做外部中断输入，则使用下面的方法调用本函数
AFIO_EXTISourceConfig（AFIO_EXTI_CH_6,AFIO_ESS_PB）;
//参数 AFIO_EXTI_CH_6 表示第 6 个 IO 口
//参数 AFIO_ESS_PB 表示 GPIOB 口

3.2.3、AFIO_GPAConfig

```
void AFIO_GPAConfig    ( u32    AFIO_PIN,
                         u32    AFIO_MODE
 )
```

Configure alternated mode of GPIOA with specified pins.
配置 GPIOA 口的指定引脚的复用模式
Parameters:
函数参数
AFIO_PIN,: This parameter can be any combination of AFIO_PIN_x where x can be 0 ~ 15.
AFIO_PIN,:这个参数可以是 AFIO_PIN_x 的组合，x 可以是 0~15.
AFIO_MODE,: This parameter can be one of the following values:
AFIO_MODE：这个参数是下面几个值中的一个
AFIO_MODE_DEFAULT：默认的 IO 功能
AFIO_MODE_1 :复用模式 1
AFIO_MODE_2 :复用模式 2

AFIO_MODE_3 :复用模式 3

Return values:

函数返回值

None

无

举例：系统需要使用串口，查阅 sim_HT32F1755-1765-2755_Datasheetv100.pdf 文件，串口的 UR1_TX，UR1_RX 引脚与 PA2 和 PA3 引脚复用。当使用串口时，需使用本函数将 PA2,PA3 配置成串口引脚。

由于系统使用 GPIOB 口，所以应该调用 AFIO_GPBConfig（）函数

函数的 u32 AFIO_PIN 参数：

PA2 是 GPIOB 口的引脚 2，AFIO_PIN 形参的实际参数是 AFIO_PIN_2

PA3 是 GPIOB 口的引脚 3，AFIO_PIN 形参的实际参数是 AFIO_PIN_3

由于 AFIO_PIN 参数可以组合，且需要设置的 PA2，PA3 的 AFIO_MODE 参数相同，所以：

组合后的实际参数是 AFIO_PIN_2| AFIO_PIN_3，注意”|”表示使用或逻辑运算。

这样一次函数调用同时设置多个引脚的复用模式。

函数的 u32 AFIO_MODE 参数：

查表得到这 2 个引脚串口模式配置都是 AF2

AFIO_MODE 的实际值是 AFIO_MODE_2

实际程序代码：

AFIO_GPAConfig（AFIO_PIN_2| AFIO_PIN_3，AFIO_MODE_2）;

GPIOB,GPIOC,GPIOD,GPIOE 端口的复用配置函数如下

```
void AFIO_GPBConfig    (      u32    AFIO_PIN,
                              u32    AFIO_MODE
      )
void AFIO_GPCConfig    (      u32 AFIO_PIN,
                              u32 AFIO_MODE
      )
void AFIO_GPDConfig    (      u32 AFIO_PIN,
                              u32 AFIO_MODE
      )
void AFIO_GPEConfig    (      u32 AFIO_PIN,
                              u32 AFIO_MODE
      )
```

这些函数与 AFIO_GPAConfig（）函数相似，本文不再详述。

3.2.4、GPIO_ClearOutBits

```
void GPIO_ClearOutBits    ( GPIO_TypeDef *    GPIOx,
                            u16    GPIO_PIN
 )
```

Clear the selected port bits of output data.

指定的端口位清零

Parameters:

参数

GPIOx,: where GPIOx is the selected GPIO from the GPIO peripheral, x can be A ~ E.

GPIOx：选择哪一个 GPIO 口，x 可以是 A~E。

GPIO_PIN,: Specify the port bit to be clear. This parameter can be any combination of GPIO_PIN_x where x can be 0 ~ 15.

GPIO_PIN：选择的 IO 口的那个引脚清零，这个参数可以是多个引脚的组合，GPIO_PIN_x 中的 x 可以是 0~15；

Return values:

返回值

None

无

GPIO_TypeDef * GPIOx 是库函数新定义的数据类型，见头文件。

举例：将 PA 口的 PA3,PA13 清零

因为选择 PA 口，所以形参 GPIOx 的实际值为 GPIOA，

PA3 是 GPIOB 口的引脚 3，AFIO_PIN 形参的实际参数是 AFIO_PIN_3

PA13 是 GPIOB 口的引脚 13，AFIO_PIN 形参的实际参数是 AFIO_PIN_13

由于 AFIO_PIN 参数可以组合，所以：

组合后的实际参数是 AFIO_PIN_3| AFIO_PIN_13

下面介绍的内容不在英汉对照。

3.2.5、GPIO_DeInit

```
void GPIO_DeInit   ( GPIO_TypeDef *   GPIOx )
```

函数功能：恢复 GPIO 的默认设置

函数参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

函数返回值：无

3.2.6、GPIO_DirectionConfig

```
void GPIO_DirectionConfig   ( GPIO_TypeDef *   GPIOx,
                              u16   GPIO_PIN,
                              GPIO_DIR_Enum   Direction
 )
```

函数功能：配置 IO 引脚的输入输出方向

函数调用参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.

Direction,: IO 引脚的方向，下面的值的 2 选 1
GPIO_DIR_IN : 指定引脚为输入方向
GPIO_DIR_OUT: 指定引脚为输出方向.
函数返回值：无
举例：将 GPIOA 的 PA2，PA9 设置为输出方向
GPIO_DirectionConfig（GPIOA，GPIO_PIN_2|GPIO_PIN_9，GPIO_DIR_OUT）;

3.2.7、GPIO_DriveConfig

```
void GPIO_DriveConfig  ( GPIO_TypeDef *  GPIOx,
  u16  GPIO_PIN,
  GPIO_DV_Enum  Drive
 )
```

函数功能：设置指定 IO 引脚的输出驱动电流
函数调用参数：
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.
Drive,: 驱动能力，为下面的参数 2 选 1:
GPIO_DV_4MA: 驱动能力 4 mA
GPIO_DV_8MA: 驱动能力 8 mA
举例：设置 GPIOA 的 PA2，PA9 为 8mA 电流输出
GPIO_DriveConfig（GPIOA，GPIO_PIN_2|GPIO_PIN_9，GPIO_DV_8MA）;

3.2.8、GPIO_InputConfig

```
void GPIO_InputConfig  ( GPIO_TypeDef *  GPIOx,
  u16  GPIO_PIN,
  ControlStatus  Cmd
 )
```

使能或禁止输入口的施密特触发输入功能
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.
Cmd,: 这个参数是 ENABLE 或 DISABLE.
返回值：无

3.2.9、GPIO_IsPinLocked

```
bool GPIO_IsPinLocked  ( GPIO_TypeDef *  GPIOx,
  u16  GPIO_PIN
 )
```

获取 GPIO 引脚的锁定状态

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.

函数返回值：

TRUE 或 FALSE

当函数指定的引脚处于锁定状态时返回值为 TRUE，f 否者返回值为 FALSE，TURE，FALSE 是库函数的预定义符号常数。

3.2.10、GPIO_IsPortLocked

```
bool GPIO_IsPortLocked   ( GPIO_TypeDef *   GPIOx )
```

获取 GPIO 端口的锁定状态

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

返回值：TRUE or FALSE

3.2.11、GPIO_OpenDrainConfig

```
void GPIO_OpenDrainConfig   ( GPIO_TypeDef *   GPIOx,
  u16   GPIO_PIN,
  ControlStatus   Cmd
 )
```

使能或禁止 GPIO 引脚的开漏输出功能

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.

Cmd,: 这个参数可以是 ENABLE 或 DISABLE.

返回值：无

3.2.12、GPIO_PinLock

```
void GPIO_PinLock   ( GPIO_TypeDef *   GPIOx,
  u16   GPIO_PIN
 )
```

功能：锁定 GPIO 引脚的配置状态

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.

返回值：无

3.2.13、GPIO_PullResistorConfig

```
void GPIO_PullResistorConfig   ( GPIO_TypeDef *    GPIOx,
  u16    GPIO_PIN,
  GPIO_PR_Enum    PR
 )
```

功能：配置 GPIO 引脚的上拉电阻功能。
参数：
Parameters:
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.
PR,: 选择上拉电阻，这个参数是下面中的一个：
GPIO_PR_UP : 内部上拉电阻
GPIO_PR_DOWN : 内部下拉电阻
GPIO_PR_DISABLE : 无上拉下拉电阻功能
返回值：无

3.2.14、GPIO_ReadInBit

```
FlagStatus GPIO_ReadInBit   ( GPIO_TypeDef *    GPIOx,
                              u16    GPIO_PIN
 )
```

读取指定 GPIO 引脚的输入数据.
参数：
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x ，x 可以是 0 ~ 15.（不能组合使用）
返回值：如果指定引脚为高电平，本函数返回 SET，否者，返回 RESET。

3.2.15、GPIO_ReadInData

```
u16 GPIO_ReadInData   ( GPIO_TypeDef *    GPIOx )
```

功能：读取 GPIO 口的输入数据
参数：
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
返回值：指定 GPIO 口的输入数据寄存器值

3.2.16、GPIO_ReadOutBit

```
FlagStatus GPIO_ReadOutBit   ( GPIO_TypeDef *   GPIOx,
  u16   GPIO_PIN
 )
```

功能：读取指定引脚的输出数据

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x ，x 可以是 0 ~ 15.（不能组合使用）

返回值：SET 或 RESET

3.2.17、GPIO_ReadOutData

```
u16 GPIO_ReadOutData   ( GPIO_TypeDef *   GPIOx )
```

Get the output data of specified GPIO port.

功能：读取指定 GPIO 口的输出数据

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

返回值：

指定 GPIO 口的输出寄存器值。

3.2.18、GPIO_SetOutBits

```
void GPIO_SetOutBits   ( GPIO_TypeDef *   GPIOx,
  u16   GPIO_PIN
 )
```

功能：置位选择的 GPIO 引脚

参数：

GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.

GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.

返回值：无

3.2.19、 GPIO_WriteOutBits

```
void GPIO_WriteOutBits   ( GPIO_TypeDef *   GPIOx,
  u16   GPIO_PIN,
```

```
  FlagStatus    Status
 )
```

功能：置位或清零指定的 GPIO 引脚.
参数：
GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
GPIO_PIN,: 选择哪一个引脚，该值是 GPIO_PIN_x 的组合 x 可以是 0 ~ 15.
Status,: 指定选择的引脚输出状态，可以是下面的值之一。
RESET : 清零
SET : 置位
返回值：无

3.2.20、GPIO_WriteOutData

```
void GPIO_WriteOutData   ( GPIO_TypeDef *    GPIOx,
  u16    Data
 )
```

功能：向指定的 GPIO 口（*GPIOx*）输出数据（*Data*）.
参数：
GPIOx,: GPIOx,: 选择哪一个 GPIO 口, x 可以是 A ~ E.
Data,: IO 口输出的数据
返回值：无