NV32F100x BOS 模块编程示例

第一章 库函数简介

1.1 库函数列表

BME_AND(ADDR)

位操作存储逻辑与（AND）32 位

BME_OR(ADDR)

位操作存储逻辑或（OR）32 位

BME_XOR(ADDR)

位操作存储逻辑异或（XOR）32 位

BME_BITFIELD_INSERT(ADDR,bit,width)

位操作存储字段插入（BFI）32 位

BME_BIT_CLEAR(ADDR,bit)

位操作存储加载：1 位加载—清零（LAC1）32 位

BME_BIT_SET(ADDR,bit)

位操作存储加载：1 位加载—置位（LAS1）32 位

BME_BITFIELD_EXTRACT(ADDR,bit,width)

位操作存储加载无符号位字段提取（UBFX）32 位

BME_AND_8b(ADDR)

位操作存储逻辑与（AND）8 位

BME_OR_8b(ADDR)

位操作存储逻辑或（OR）8 位

BME_XOR_8b(ADDR)

位操作存储逻辑异或（XOR）8 位

BME_BITFIELD_INSERT_8b(ADDR,bit,width)

位操作存储字段插入（BFI）8 位

BME_BIT_CLEAR_8b(ADDR,bit)

位操作存储加载：1 位加载—清零（LAC1）8 位

BME_BIT_SET_8b(ADDR,bit)

位操作存储加载：1 位加载—置位（LAS1）8 位

BME_BITFIELD_EXTRACT_8b(ADDR,bit,width)

位操作存储加载无符号位字段提取（UBFX）8 位

BME_AND_16b(ADDR)

位操作存储逻辑与（AND）16 位

BME_OR_16b(ADDR)

位操作存储逻辑或（OR）16 位

BME_XOR_16b(ADDR)

位操作存储逻辑异或（XOR）16 位

BME_BITFIELD_INSERT_16b(ADDR,bit,width)

位操作存储字段插入（BFI）16 位

BME_BIT_CLEAR_16b(ADDR,bit)

位操作存储加载：1 位加载—清零（LAC1）16 位

BME_BIT_SET_16b(ADDR,bit)

位操作存储加载：1 位加载—置位（LAS1）16 位

BME_BITFIELD_EXTRACT_16b(ADDR,bit,width)

位操作存储加载无符号位字段提取（UBFX）16 位

1.2 BOS 的主要特性

*针对选定地址空间进行的位操作存储的轻量级实施

*将附加访问语义符号编码到参考地址中

*位于处理器内核和开关主端口之间

*符合 AHB 系统总线协议的两级流水线设计

*可将非位操作存储访问组合传输至从设备总线控制器

*可将来自处理器内核的位操作存储加载和存储转换为基元读取-修改-写入

*位操作存储加载支持无符号位字段提取、加载和{置位，清零} 1 位操作

*位操作存储支持位字段插入、逻辑“与”、“或”和“异或”操作

*支持字节、半字和字大小的位操作存储

*支持在 AHB 输出总线上执行最少的信号切换，以降低功耗

1.3 BOS 使用说明

*选择 BOS 位操作的功能（AND、OR、XOR 以及一个位字段插入）
*根据要操作寄存器的地址生成 BOS 相应操作功能的操作地址，实现对寄存器操作

详细 BOS 位操作请参阅读参考手册

第二章 BOS 位操作存储

BOS 位操作存储支持的功能包括三个逻辑运算符（AND、OR、XOR）以及一个位字段插入，详细内容请参阅参考手册。

2.1 位操作存储逻辑与（AND）

该命令执行参考存储器位置的基元读取-修改-写入。数据尺寸由写入操作定义，可以是字节（8 位）、半字（16 位）或字（32 位）。进行字节和半字传输时，内核执行所需的写入数据通道复制操作。详细内容请参与参考手册。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ioandb	0	*	*	0	0	1	-	-	-	-	-	-	mem-addr																			
ioandh	0	*	*	0	0	1	-	-	-	-	-	-	mem-addr																		0	
ioandw	0	*	*	0	0	1	-	-	-	-	-	-	mem-addr																	0	0	

图 2-1 位操作存储地址：逻辑与(AND)

其中 addr[30:29] = 01 (SRAM_U)，addr[30:29] = 10（外设），addr[28:26]= 001（指定 AND 操作），mem_addr[19:0]指定空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”

函数名	BME_AND
函数原形	BME_AND(ADDR)
功能描述	对寄存器数据执行与操作
输入参数	要操作寄存器地址
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， BME_AND(&GPIOA->PDOR) = 0x02;

```
/*******************************************************************************!
    *
    * @ 概要                生成 BOS AND 操作地址 (硬件编码地址 32 位).
    * @ 参数【输入】        ADDR        32 位地址
    * @ 返回硬件编码的 32 位地址
    *
*******************************************************************************/
```

```
#define BME_AND(ADDR)          (*(volatile uint32_t *)(((uint32_t)ADDR) | (BME_OPCODE_AND<<26)))
```

2.2 位操作存储逻辑或(OR)

该命令执行参考存储器位置的基元读取-修改-写入。数据尺寸由写入操作定义，可以是字节（8 位）、半字（16 位）或字（32 位）。进行字节和半字传输时，内核执行所需的写入数据通道复制操作。详细内容请参阅参考手册。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ioorb	0	*	*	0	1	0	-	-	-	-	-	-	mem-addr																			
Ioorh	0	*	*	0	1	0	-	-	-	-	-	-	mem-addr																			0
Ioorw	0	*	*	0	1	0	-	-	-	-	-	-	mem-addr																		0	0

图 2-2 位操作存储地址存储：逻辑或(OR)

其中 addr[30:29] = 01 (SRAM_U)，addr[30:29] = 10（外设），addr[28:26]= 010（指定 OR 操作），mem_addr[19:0]指定空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”

函数名	BME_OR
函数原形	BME_OR(ADDR)
功能描述	对寄存器数据执行或操作
输入参数	要操作寄存器地址
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， BME_OR(&GPIOA->PDOR) = 0x02;

```
/*******************************************************************************//*!
    *
    * @ 概要 生成 BOS   OR 操作地址 (硬件编码地址 32 位).
    * @ 参数【输入】    ADDR       32 位地址
    * @ 返回   硬件编码的 32 位地址
    *
*******************************************************************************/
#define BME_OR(ADDR)           (*(volatile uint32_t *)(((uint32_t)ADDR) | (BME_OPCODE_OR<<26)))
```

2.3 位操作存储逻辑异或(XOR)

该命令执行参考存储器位置的基元读取-修改-写入。数据尺寸由写入操作定义，可以是字节（8 位）、半字（16 位）或字（32 位）。进行字节和半字传输时，内核执行所需的写入数据通道复制操作。详细内容

请参阅参考手册。

图 2-3 位操作存储地址存储：逻辑异或(XOR)

其中 addr[30:29] = 01(SRAM_U)，addr[30:29] =10（外设），addr[28:26]= 011（指定 XOR 操作），mem_addr[19:0]指定外设空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”

函数名	BME_XOR
函数原形	BME_XOR(ADDR)
功能描述	对寄存器数据执行与操作
输入参数	要操作寄存器地址
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， BME_XOR(&GPIOA->PDOR) = 0x02;

```
/*******************************************************************************//*!
    *
    * @ 概要 生成 BOS  XOR 操作地址 (硬件编码地址 32 位).
    * @ 参数【输入】    ADDR      32 位地址
    * @ 硬件编码的 32 位地址
    *
*******************************************************************************/
#define BME_XOR(ADDR)          (*(volatile uint32_t *)(((uint32_t)ADDR) | (BME_OPCODE_XOR<<26)))
```

2.4 位操作存储位字段插入(BFI)

该命令使用一个基元读取-修改-写入序列，将由最低有效位(b)和位字段宽度(w+1)定义的写数据操作数中所含的位字段插入由与存储指令有关的访问尺寸定义的存储器“容器”中。数据尺寸由写入操作定义，可以是字节（8 位）、半字（16 位）或字（32 位）。详细内容请参阅参考手册。

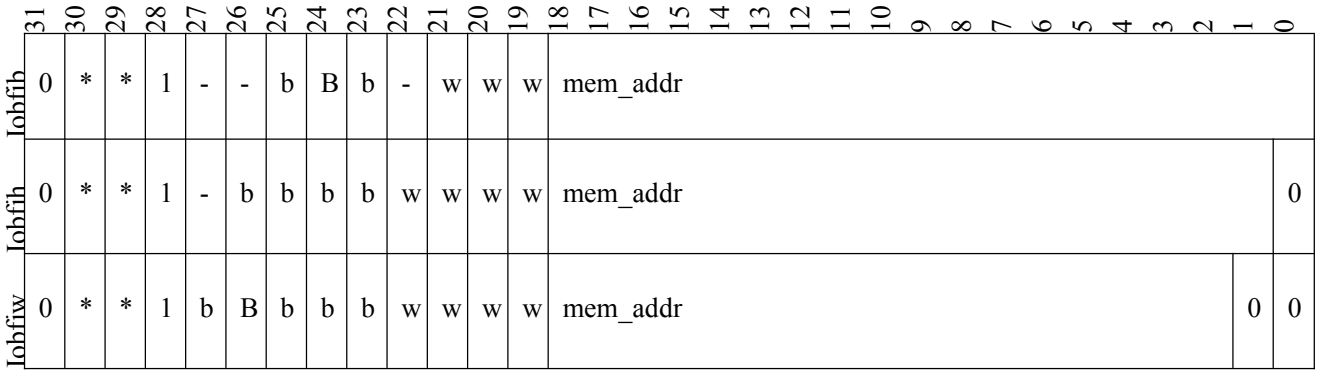

图 2-4 位操作存储地址：位字段插入

其中，addr[30:29] = 01 (SRAM_U)，addr[30:29] =10（外设），addr[28] = 1（表示 BFI 操作），addr[27:23] = "b"（LSB 标识符），addr[22:19] = "w"（位字段宽度减 1 标识符），addr[18:0]指定外设空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”。注意，与其他位操作存储保存操作不同，BFI 使用 addr[19]作为"w"指示符中的最低有效位，而非地址位

函数名	BME_BITFIELD_INSERT
函数原形	BME_BITFIELD_INSERT(ADDR,bit,width)
功能描述	对寄存器执行位字段的插入
输入参数	要操作寄存器地址，位数、域宽
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， 位数、域宽 BME_BITFIELD_INSERT(u32Addr,16,4) = u32Data

```
/*******************************************************************************//*!
    *
    * @ 概要  生成 BOS 位域插入操作地址 (硬件编码地址 32 位).
    *
    * @ 参数【输入】     ADDR   32 位地址
    *
    * @ 参数【输入】   bit       位数 ，  0-基址
    *
    * @ 参数【输入】  width   位域宽, 1-基址.
    *
    * @ 返回 32 位硬件编码地址
    *
******** ******************************************************************************/
    #define BME_BITFIELD_INSERT(ADDR,bit,width)            (*(volatile uint32_t *)(((uint32_t)ADDR)     \
                                                | (BME_OPCODE_BITFIELD <<26)   \
                                                | ((bit)<<23) | ((width-1))<<19))
```

2.5 位操作存储加载：1 位加载-清零（LAC1）

该命令将 LSB 位(b)定义的 1 位字段加载至内核通用目标寄存器(Rt)，并在执行基元读取-修改-写入序列后清零存储器空间中的位。从存储器地址中提取的 1 位数据字段在返回至内核的操作数中右对齐和零填充。数据尺寸由读取操作指定，可以是字节（8 位）、半字（16 位）或字（32 位）。详细内容请参阅参考手册。

图 2-5 位操作存储加载地址：1 位加载－清零

其中：addr[30:29] = 01 (SRAM_U)，addr[30:29] = 10（ 外设），addr[28:26]= 010（指定 1 位加载－清零操作），addr[25:21] = "b"（位标识符），mem_addr[19:0]指定空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”。

函数名	BME_BIT_CLEAR
函数原形	BME_BIT_CLEAR(ADDR,bit)
功能描述	对寄存器执行清零操作
输入参数	要操作寄存器地址，位数、域宽
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， 位数、域宽 BME_BIT_CLEAR(u32Addr,16,4) = u32Data

```
/*******************************************************************************//*!
    *
    * @ 概要   生成 BOS 位清零操作地址 (硬件编码地址 32 位).
    *
    * @ 参数【输入】    ADDR   32 位地址
    * @ 参数【输入】 bit       位数 ，  0-基址
    * @ 参数【输入】   width   位域宽, 1-基址.
    *
    * @ 返回 32 位硬件编码地址
    *
*******************************************************************************/
```

```
#define BME_BIT_CLEAR(ADDR,bit)          (*(volatile uint32_t *)(((uint32_t)ADDR)    \
                                         | (BME_OPCODE_BIT_CLEAR <<26)   \
                                         | ((bit)<<21)))
```

2.6 位操作存储加载：1 位加载-置位(LAS1)

该命令将 LSB 位(b)定义的 1 位字段加载至内核通用目标寄存器(Rt)，并在执行基元读取-修改-写入序列后置位存储器空间中的位。从存储器地址中提取的 1 位数据字段在返回至内核的操作数中右对齐和零填充。数据尺寸由读取操作指定，可以是字节（8 位）、半字（16 位）或字（32 位）。详细内容请参阅参考手册

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Iolaslb	0	*	*	0	1	1	-	-	b	b	b	-	mem_addr																			
Iolaslh	0	*	*	0	1	1	-	b	b	b	B	-	mem_addr																			0
Iolaslw	0	*	*	0	1	1	b	b	b	b	B	-	mem_addr																		0	0

图 2-6 位操作存储加载地址：1 位加载－置位

其中，addr[30:29] = 01 (SRAM_U)，addr[30:29] = 10（外设），addr[28:26] = 011（指定 1 位加载－置位操作），addr[25:21] = "b"（位标识符），mem_addr[19:0]指定空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000）。"-"表示该地址位为“无关位”。

函数名	BME_BIT_SET
函数原形	BME_BIT_SET(ADDR,bit)
功能描述	对寄存器执行置位操作
输入参数	要操作寄存器地址，位数、域宽
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， 位数、域宽 BME_BIT_SET(u32Addr,16,4) = u32Data

```
/*******************************************************************************//*!
    *
    * @ 概要   生成 BOS 位置位操作地址 (硬件编码地址 32 位).
    *
    * @ 参数【输入】    ADDR   32 位地址
    * @ 参数【输入】    bit      位数 ，  0-基址
    * @ 参数【输入】   width   位域宽, 1-基址.
    *
```

```
    * @ 返回 32 位硬件编码地址
    *
****** ********************************************************************************/
    #define BME_BIT_SET(ADDR,bit)          (*(volatile uint32_t *)(((uint32_t)ADDR)     \
                                          | (BME_OPCODE_BIT_SET <<26)    \
                                          | ((bit)<<21)))
```

2.7 位操作存储加载无符号位字段提取（UBFX）

该命令使用一个双周期读取序列，从与加载指令有关的访问尺寸所定义的存储器“容器”中提取 LSB 位(b)定义的位字节和位字段宽度(w+1)。从存储器地址中提取的位字段在返回至内核的操作数中右对齐和零填充。如前文所述，这是唯一一个不执行存储器写入的位操作存储，即 UBFX 仅执行读取操作。数据尺寸由写入操作定义，可以是字节（8 位）、半字（16 位）或字（32 位）。详细内容请参阅参考手册

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ioubfxb	0	*	*	1	-	-	b	B	b	-	w	w	w	mem_addr																		
ioubfxh	0	*	*	1	-	b	b	b	b	w	w	w	w	mem_addr																		0
ioubfxw	0	*	*	1	b	b	b	b	b	w	w	w	w	mem_addr																	0	0

图 2-7 位操作存储加载地址：无符号位字段提取

其中：addr[30:29] = 01 (SRAM_U)，addr[30:29] = 10（外设），addr[28]= 1（指定无符号位字段提取操作），addr[27:23] = "b"（LSB 标识符），addr[22:19] ="w"（位字段宽度减 1 标识符），mem_addr[18:0]指定空间偏移地址（对于 SRAM_U，基地址为 0x2000_0000；对于外设，基地址为 0x4000_0000。"-"表示该地址位为“无关位”。

函数名	BME_BITFIELD_EXTRACT
函数原形	BME_BITFIELD_EXTRACT(ADDR,bit,width)
功能描述	对寄存器执行数据提取操作
输入参数	要操作寄存器地址，位数、域宽
输出参数	无
返回值	无
先决条件	无
函数使用实例	先设置寄存器地址， 位数、域宽 BME_BITFIELD_EXTRACT(u32Addr,16,4) = u32Data

```
/*********************************************************************************************//*!
   *
```

```
  * @概要   生成 BOS 位域提取操作地址 (硬件编码地址 32 位).
  *
  * @ 参数【输入】   ADDR   32 位地址
  * @ 参数【输入】   bit      位数 ，  0-基址
  * @ 参数【输入】   width   位域宽, 1-基址.
  *
  * @ 返回 32 位硬件编码地址
  *
*******************************************************************************/
#define BME_BITFIELD_EXTRACT(ADDR,bit,width)          (*(volatile uint32_t *)(((uint32_t)ADDR)      \
                                                    | (BME_OPCODE_BITFIELD <<26)   \
                                                    | ((bit)<<23) | ((width-1))<<19))
```

第三章 样例程序

BME_demo

```
/*******************************************************************************************************
*
*    该例程分别记录使用常规 C 语言执行逻辑操作 CPU 执行时间、使用 BOS 执行逻辑操作 CPU 执行时间
*    使用常规 C 语言执行位域操作 CPU 执行时间、使用 BOS 执行位域操作的 CPU 执行时间。通过比较可
*    看出 BOS 模块的可降低 CPU 执行时间。
*
*******************************************************************************************************/
#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "systick.h"
#include "bme.h"
#include "sysinit.h"
#include "start.h"

#define GPIO_ALIAS_OFFSET            0x000F0000L
  /*!< GPIO 端口数据数据寄存基地址别名，被用 BOS BFI/UBFX 函数   */
#define GPIOB_PDOR_ALIAS             (((uint32_t)&GPIOB->PDOR)-GPIO_ALIAS_OFFSET)
 /*!< GPIO 端口数据数据寄存基地址别名，被用 BOS BFI/UBFX 函数    */

int main (void);
void RTC_Task(void);
uint32_t BME_LogicOPwithC(void);
uint32_t BME_LogicOPwithBME(void);
```

```
uint32_t BME_BFIwithC(uint32_t *pAddr, uint8_t u8BitPos, uint8_t u8FieldWidth, uint32_t u32Data);
uint32_t BME_BFIwithBME(void);

#ifdef __GNUC__
#define __ramfunc __attribute__((section (".data")))
#endif

int main (void)
{
    /* 系统初始化 */
    sysinit();
    cpu_identify();

    ICS_ConfigType   sICSConfig;
    RTC_ConfigType   sRTCConfig;
    RTC_ConfigType   *pRTCConfig = &sRTCConfig;
    UART_ConfigType sConfig;

    printf("\r\nRunning the BME_demo project.\r\n");
#if defined(__ICCARM__)
    printf("Build by IAR\r\n");
#elif defined(__GNUC__)
    printf("Build by GUNC\r\n");
#elif defined(__CC_ARM)
    printf("Build by MDK\r\n");
#else
    printf("Unrecognized compiler!\r\n");
#endif
    LED0_Init();
    LED2_Init();

    /* 配置 RTC 配置结构体，使其每隔 1s 产生一次中断 */
    pRTCConfig->u16ModuloValue = 9;
    pRTCConfig->bInterruptEn     = RTC_INTERRUPT_ENABLE;      /* 使能中断 */
    pRTCConfig->bClockSource     = RTC_CLKSRC_1KHZ;           /*时钟源 1khz*/
    pRTCConfig->bClockPresaler = RTC_CLK_PRESCALER_100;     /* 时钟分频系数 100*/

    RTC_SetCallback(RTC_Task);
    RTC_Init(pRTCConfig);

    printf("\nIt is in FEE mode now,");
    UART_WaitTxComplete(TERM_PORT);

    /* 将时钟从 FEE 模式切换到 FEI 模式*/
```

```
    sICSConfig.u32ClkFreq = 10000;
    ICS_SwitchMode(FEE,FEI, &sICSConfig);

    /* 由于总线时钟变为 20MHz 从新设置 UART 模块*/
    sConfig.u32SysClkHz = 48000000L;
    sConfig.u32Baudrate    = UART_PRINT_BITRATE;

    UART_Init (TERM_PORT, &sConfig);

    printf("switch to FEI mode.\n");

    OSC_Enable();   //使能晶振

    /* 开始使用常规 C 和 BOS 进行位操作 */
    printf("Logic operation in C takes %d ticks!\r\n", BME_LogicOPwithC());
    printf("Logic operation with BME takes %d ticks!\r\n", BME_LogicOPwithBME());
    /*!
     * 设置 PG0~PG3 引脚为输出
     */
    GPIOB->PDDR |= (0xF << 16);                 /* 设置 PG0~PG3 为输出*/
    GPIOB->PIDR &= ~(0xF << 16);        /* 注；为了能从端口读取到正确的数据 PIDR 位必须清零 */

 printf("Bit field operation in C takes %d ticks!\r\n", BME_BFIwithC((uint32_t*)&GPIOB->PDOR,16,4-1,
5<<16));
    printf("Bit field operation with BME takes %d ticks!\r\n", BME_BFIwithBME());
    printf("Test completed!\n");
    while(1)
    {
    }
}
/*****************************************************************************//*!
*
* @概要  使用常规 C 语言执行逻辑操作
*
* @ 无参数
*
* @ 返回逻辑操作 CPU 执行的时间
*
*****************************************************************************/
#if (defined(__ICCARM__) || defined(__GNUC__))
__ramfunc uint32_t BME_LogicOPwithC(void)
#elif defined(__CC_ARM)
uint32_t BME_LogicOPwithC(void)
#endif
```

```
{
    uint32_t    u32PortVal = 0;
    uint32_t    u32LogicOPTicks;
    /*! 配置 PA1 作为输出引脚  */
    GPIOA->PDDR |= 0x02;
    GPIOA->PIDR &= ~0x02; /* 注；为了能从端口读取到正确的数据 PIDR 位必须清零*/
    /* 初始化系统时钟和逻辑操作的计数时间*/
    SysTick->CTRL &= ~(SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    SysTick->VAL = 0x0; /* 清除寄存器当前值*/
    SysTick->LOAD = 0x00FFFFFF;
    SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    /* exe: */
  /* 使用常规的方法读取寄存器、修改数值、写入寄存器切换 GPIO     */
    u32PortVal = GPIOA->PDOR;
    u32PortVal ^= 0x02;
    GPIOA->PDOR = u32PortVal;

    u32PortVal = GPIOA->PDOR;
    u32PortVal ^= 0x02;
    GPIOA->PDOR = u32PortVal;

    u32PortVal = GPIOA->PDOR;
    u32PortVal ^= 0x02;
    GPIOA->PDOR = u32PortVal;

    u32PortVal = GPIOA->PDOR;
    u32PortVal ^= 0x02;
    GPIOA->PDOR = u32PortVal;

    u32PortVal = GPIOA->PDOR;
    u32PortVal ^= 0x02;
    GPIOA->PDOR = u32PortVal;
    /* 测量逻辑操作 CPU 执行时间*/
    u32LogicOPTicks = SysTick->VAL;
    return (SysTick->LOAD - u32LogicOPTicks);
}
/*******************************************************************************//*!
*
* @概要  通过 B0S 执行逻辑操作
*
* @ 无参数
*
* @ 返回逻辑操作 CPU 执行的时间
*
```

```
*****************************************************************************/
#if (defined(__ICCARM__) || defined(__GNUC__))
__ramfunc uint32_t BME_LogicOPwithBME(void)
#elif defined(__CC_ARM)
uint32_t BME_LogicOPwithBME(void)
#endif
{
    uint32_t      u32LogicOPTicks;
    /*!
    * 配置 PA1 作为输出端口
    */
    GPIOA->PDDR |= 0x02;
    GPIOA->PIDR &= ~0x02;              /* 注；为了能从端口读取到正确的位置 PIDR 位必须清零*/

    /*初始化系统时钟和逻辑操作的计数时间 */
    SysTick->CTRL &= ~(SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    SysTick->VAL = 0x0;   /* 清除寄存器当前值 */
    SysTick->LOAD = 0x00FFFFFF;
    SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    /* exe: */
    BME_XOR(&GPIOA->PDOR) = 0x02;
    BME_XOR(&GPIOA->PDOR) = 0x02;
    BME_XOR(&GPIOA->PDOR) = 0x02;
    BME_XOR(&GPIOA->PDOR) = 0x02;
    BME_XOR(&GPIOA->PDOR) = 0x02;

    /*测量逻辑操作 CPU 执行的时间 */
    u32LogicOPTicks = SysTick->VAL;
    return (SysTick->LOAD - u32LogicOPTicks);
}
/*****************************************************************************//*!
*
* @概要  使用常规 C 语言执行位域操作.
*
* @ 参数【输入】 pAddr              指向 32 位数据的目标地址
* @ 参数【输入】 u8BitPos           被操作的位域地址的 32 位数据
* @ 参数【输入】 u8FieldWidth       被代替的 32 位数据的区域宽度减一
* @ 参数【输入】 u32Data          32 位数据包含插入到相应的位域的 32 位数据的位字段
*
* @返回位域操作 CPU 执行时间
*
*****************************************************************************/
#if (defined(__ICCARM__) || defined(__GNUC__))
```

```
__ramfunc uint32_t BME_BFIwithC(uint32_t *pAddr, uint8_t u8BitPos, uint8_t u8FieldWidth, uint32_t u32Data)
#elif defined(__CC_ARM)
uint32_t BME_BFIwithC(uint32_t *pAddr, uint8_t u8BitPos, uint8_t u8FieldWidth, uint32_t u32Data)
#endif
{
    uint32_t    u32RegVal;
    uint32_t    u32Mask;
    uint32_t    u32LogicOPTicks;

    /* 初始化系统时钟和位域操作的计数时间*/
    SysTick->CTRL &= ~(SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    SysTick->VAL = 0x0;   /* 清除寄存器当前值 */
    SysTick->LOAD = 0x00FFFFFF;
    SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    /* exe: */
    u32RegVal = *pAddr;
    u32Mask = ((1 << (u8FieldWidth+1)) - 1) << u8BitPos;
    u32RegVal = (u32RegVal & ~u32Mask)|((u32Data) & u32Mask);
    *pAddr = u32RegVal;
    /* 测量位域操作 CPU 执行时间*/
    u32LogicOPTicks = SysTick->VAL;
    return (SysTick->LOAD - u32LogicOPTicks);
}

/*******************************************************************************//*!
*
* 概要  使用 BOS 执行位域操作
*       执行位域操作模拟 GPIO 掩码
*
* 返回 CPU 执行时间
*
*******************************************************************************/
#if (defined(__ICCARM__) || defined(__GNUC__))
__ramfunc uint32_t BME_BFIwithBME(void)
#elif defined(__CC_ARM)
uint32_t BME_BFIwithBME(void)
#endif
{
    uint32_t    u32LogicOPTicks;
    uint32_t    u32Data = (0x5 << 16);
    uint32_t    u32Addr = GPIOB_PDOR_ALIAS;
    /* 配置 PG0~PG3 作为输出引脚*/
    GPIOB->PDDR |= (0xF << 16);                 /*配置 PG0~PG3 作为输出引脚 */
```

```
    GPIOB->PDOR =    0;                          /* 输出 0 到 PG0~PG3 端口 */
    GPIOB->PIDR &= ~(0xF << 16);                 /*注；为了能从端口读取到正确的数值 PIDR 位必须清零 */

    /* 初始化系统时钟和位域操作的时间 */
    SysTick->CTRL &= ~(SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    SysTick->VAL = 0x0;    /* 清除寄存器当前值*/
    SysTick->LOAD = 0x00FFFFFF;
    SysTick->CTRL |= (SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_ENABLE_Msk);
    /* exe: */
    BME_BITFIELD_INSERT(u32Addr,16,4) =    u32Data;    /* write 5 to bit 19..16 */
    /*测量位域操作 CPU 执行时间 */
    u32LogicOPTicks = SysTick->VAL;
    return (SysTick->LOAD - u32LogicOPTicks);
}
/*****************************************************************************//*!
*
*RTC 任务函数
*闪烁 LED 灯
*
*****************************************************************************/
void RTC_Task(void)
{
    /* 闪烁 LED1 */
    LED0_Toggle();
}
/*****************************************************************************/
```