AN0104

应用笔记

AT32外部中断/事件EXINT使用指南

前言

这篇应用笔记介绍AT32系列MCU的EXINT功能及其固件驱动程序API，并对BSP例程的软件设计加以说明，同时演示使用方法并展示实验效果，供用户参考。

注：本应用笔记对应的代码是基于雅特力提供的V2.x.x板级支持包（BSP）而开发，对于其他版本BSP，需要注意使用上的区别。

支持型号列表：

支持型号	AT32 全系列

目录

1 EXINT 概述 6

2 EXINT 功能描述 7

3 EXINT 配置流程 9

3.1 EXINT 寄存器描述 9

3.2 EXINT 中断初始化流程 9

3.3 中断清除 9

4 EXINT 固件驱动程序 API 10

4.1 EXINT 复位 10

4.2 EXINT 默认参数配置 10

4.3 EXINT 初始化 11

4.4 EXINT 标志位清除 12

4.5 EXINT 标志位获取 12

4.6 EXINT 软件触发 12

4.7 EXINT 中断使能 13

4.8 EXINT 事件使能 13

5 EXINT 中断向量 14

6 EXINT 案例 16

6.1 案例 1--外部中断配置 16

6.1.1 功能简介 16

6.1.2 资源准备 16

6.1.3 软件设计 16

6.1.4 实验效果 19

6.2 案例 2--EXINT 软件触发 20

6.2.1 功能简介 20

6.2.2 资源准备 20
6.2.3 软件设计 20
6.2.4 实验效果 22

7 文档版本历史 23

表目录

表 1. AT32 系列事件唤醒源表 7
表 2. EXINT 寄存器映像和复位值 9
表 3. AT32F403A/407/407A 的向量表 14
表 4. 文档版本历史 23

图目录

图 1. AT32F407 的 EXINT 框图 7

1 EXINT 概述

EXINT 共计有 16+N 条中断线 EXINT_LINE，每条中断线均支持通过边沿检测触发和软件触发来产生中断或事件。其中，16 为中断线 0~15 映射 GPIOx 的 16 个端口，N 为不同型号 AT32 独有的事件唤醒源，在 EXINT 功能描述章节的表 1 进行详细描述。

EXINT 可以根据软件配置，独立的使能或禁止中断或事件，并采取不同的边沿检测方式（检测上升沿/检测下降沿/同时检测上升沿和下降沿） 以及触发方式（边沿检测触发/软件触发/边沿检测和软件同时触发）响应触发源独立的产生中断或事件。

EXINT 主要特性：

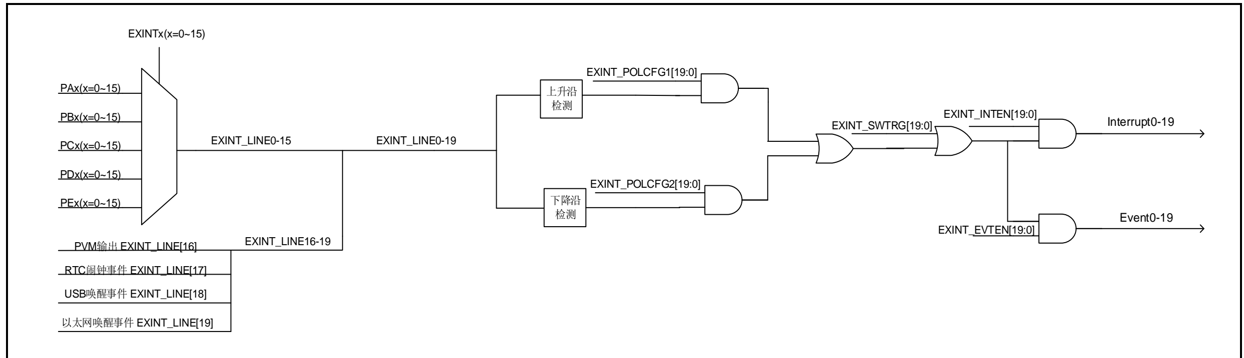
- 中断线 0~15 所映射的 GPIO 可以独立的配置
- 每个中断线都有独立的触发方式选择
- 每个中断都有独立的使能位
- 每个事件都有独立的使能位
- 可独立产生和清除的软件触发
- 每个中断都有独立的状态位
- 每个中断都可以被独立的清除

2 EXINT 功能描述

来自 GPIO 的 16 个中断源可以通过软件编程配置。对于 AT32F403A/407/413/415，配置 IOMUX 的复用外部中断配置寄存器 x（IOMUX_EXINTCx），对于 AT32F421/407/425/435/437，配置 SCFG 外部中断配置寄存器 x（SCFG_EXINTCx）。

需要注意的是这些输入源是互斥的，例如 EXINT_LINE0 只能选择 PA0/PB0/PC0/PD0…中的某一个，而不能同时选择 PA0 和 PB0 作为输入源。

图 1. AT32F407 的 EXINT 框图



对于不同型号 AT32 独有的 N 个事件唤醒源，参考上图 1，以 AT32F407 为例进行说明。AT32F407 的 EXINT 共计有 20 条中断线 EXINT_LINE[19:0]，其中 EXINT_LINE[15:0]为 GPIO 的 16 个外部中断源，EXINT_LINE[16]为 PVM 输出，EXINT_LINE[17]为 RTC 闹钟事件，EXINT_LINE[18]为 USB 唤醒事件，EXINT_LINE[19]为以太网唤醒事件。

各型号 AT32 的 EXINT_LINE[15:0] 以外的事件唤醒源请参考对应技术手册 RM，简单汇总至下表：

表 1. AT32 系列事件唤醒源表

型号	中断线 EXINT_LINE[x]	内部事件唤醒源
AT32F403A/AT32F407	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	RTC 闹钟事件
	EXINT_LINE[18]	USB 唤醒事件
	EXINT_LINE[19]	以太网唤醒事件（仅 407）
AT32F413	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	RTC 闹钟事件
	EXINT_LINE[18]	USB 唤醒事件
AT32F415	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	ERTC 闹钟事件
	EXINT_LINE[18]	OTGFS 唤醒事件
	EXINT_LINE[19]	CMP1 事件
	EXINT_LINE[20]	CMP2 事件
	EXINT_LINE[21]	ERTC 入侵和时间戳事件
	EXINT_LINE[22]	ERTC 唤醒事件

型号	中断线 EXINT_LINE[x]	内部事件唤醒源
AT32F421	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	ERTC 闹钟事件
	EXINT_LINE[19]	ERTC 入侵和时间戳事件
	EXINT_LINE[21]	COMP 事件
AT32F425	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	ERTC 闹钟事件
	EXINT_LINE[18]	OTGFS 唤醒事件
	EXINT_LINE[19]	ERTC 入侵和时间戳事件
	EXINT_LINE[20]	ERTC 唤醒事件
AT32F435/ AT32F437	EXINT_LINE[16]	PVM 输出
	EXINT_LINE[17]	ERTC 闹钟事件
	EXINT_LINE[18]	OTGFS1 唤醒事件
	EXINT_LINE[19]	以太网唤醒事件（仅 437）
	EXINT_LINE[20]	OTGFS2 唤醒事件
	EXINT_LINE[21]	ERTC 入侵和时间戳事件
	EXINT_LINE[22]	ERTC 唤醒事件

3 EXINT 配置流程

3.1 EXINT 寄存器描述

- EXINT 支持多种边沿检测方式，每条中断线可以通过软件编程配置极性配置寄存器1（EXINT_POLCFG1）独立的选择禁止或使能上升沿检测，通过配置极性配置寄存器2（EXINT_POLCFG2）独立的选择禁止或使能下降沿检测，也可以同时配置上升沿和下降沿检测，中断线上检测到的有效边沿触发可以用于产生事件或中断。
- EXINT支持独立的软件触发产生中断或事件，即除了来自中断线上的有效边沿外，用户可以通过软件编程配置软件触发寄存器（EXINT_SWTRG）对应位来产生中断或事件。
- EXINT具备独立的中断和事件使能位，用户可以通过配置中断使能寄存器（EXINT_INTEN）和事件使能寄存器（EXINT_EVTEN）来使能或关闭对应的中断或事件，这意味着无论是通过边沿检测还是软件触发产生中断或事件，都需要提前使能对应的中断或事件。
- EXINT具备独立的中断状态位，用户可以通过中断状态寄存器（EXINT_INTSTS）读取对应的中断状态并通过对该寄存器相应位**写1来清除**已置位的状态标志。

表 2. EXINT 寄存器映像和复位值

寄存器简称	基址偏移量	复位值
中断使能寄存器（EXINT_INTEN）	0x00	0x0000 0000
事件使能寄存器（EXINT_EVTEN）	0x04	0x0000 0000
极性配置寄存器 1（EXINT_ POLCFG1）	0x08	0x0000 0000
极性配置寄存器 2（EXINT_ POLCFG2）	0x0C	0x0000 0000
软件触发寄存器（EXINT_ SWTRG）	0x10	0x0000 0000
中断状态寄存器（EXINT_ INTSTS）	0x14	0x0000 0000

3.2 EXINT 中断初始化流程

- 如果使用GPIO作为中断源，需要配置复用外部中断配置寄存器 x（IOMUX_EXINTCx）或SCFG外部中断配置寄存器x（SCFG_EXINTCx）
- 选择触发方式，即配置极性配置寄存器1（EXINT_POLCFG1）和极性配置寄存器2（EXINT_POLCFG2）
- 使能中断或事件，即配置中断使能寄存器（EXINT_INTEN）或事件使能寄存器（EXINT_EVTEN）
- 如果使用软件触发产生中断，还需配置软件触发寄存器（EXINT_SWTRG）产生软件触发

3.3 中断清除

- 对中断状态寄存器（EXINT_INTSTS）对应位写 1 来清除已产生的中断标志，该操作会同步清除软件触发寄存器（EXINT_SWTRG） 中的对应位。

4 EXINT 固件驱动程序 API

Artery 提供的固件驱动程序包含了一系列固件函数来管理 EXINT 的下列功能：

- EXINT复位
- EXINT默认参数配置
- EXINT初始化
- EXINT标志位清除
- EXINT标志位获取
- EXINT软件触发
- EXINT中断使能
- EXINT事件使能

4.1 EXINT 复位

void exint_reset(void);函数可以复位所有的EXINT寄存器，如3.3中断清除描述，对中断状态寄存器（EXINT_ INTSTS）写1来清除EXINT_INTSTS和EXINT_SWTRG寄存器。

```
void exint_reset(void)
{
  EXINT->inten = 0x00000000;
  EXINT->polcfg1 = 0x00000000;
  EXINT->polcfg2 = 0x00000000;
  EXINT->evten  = 0x00000000;
  EXINT->intsts = 0x000FFFFF;
}
```

4.2 EXINT 默认参数配置

void exint_default_para_init(exint_init_type *exint_struct);函数可以禁用全部EXINT_LINE，完成默认参数初始化配置。

```
void exint_default_para_init(exint_init_type *exint_struct)
{
  exint_struct->line_enable = FALSE;
  exint_struct->line_select = EXINT_LINE_NONE;
  exint_struct->line_polarity = EXINT_TRIGGER_FALLING_EDGE;
  exint_struct->line_mode = EXINT_LINE_EVENT;
}
```

4.3 EXINT 初始化

void exint_init(exint_init_type *exint_struct);函数可以完成exint_struct结构体的参数初始化配置，包括事件/中断使能，EXINT_LINE_x选择，极性配置（上升沿/下降沿/双边沿触发），调用该函数前需要先初始化exint_struct结构体成员，参考6.1.3 exint_line0_config();描述。

```
void exint_init(exint_init_type *exint_struct)
{
  uint32_t line_index = 0;
  line_index = exint_struct->line_select;
  EXINT->inten &= ~line_index;
  EXINT->evten &= ~line_index;
  if(exint_struct->line_enable != FALSE)
  {
    if(exint_struct->line_mode == EXINT_LINE_INTERRUPUT)
    {
      EXINT->inten |= line_index;
    }
    else
    {
      EXINT->evten |= line_index;
    }
    EXINT->polcfg1 &= ~line_index;
    EXINT->polcfg2 &= ~line_index;
    if(exint_struct->line_polarity == EXINT_TRIGGER_RISING_EDGE)
    {
      EXINT->polcfg1 |= line_index;
    }
    else if(exint_struct->line_polarity == EXINT_TRIGGER_FALLING_EDGE)
    {
      EXINT->polcfg2 |= line_index;
    }
    else
    {
      EXINT->polcfg1 |= line_index;
      EXINT->polcfg2 |= line_index;
    }
  }
```

```
}
```

4.4 EXINT 标志位清除

void exint_flag_clear(uint32_t exint_line);函数可以清除指定EXINT_LINE_x的中断标志位。

```
void exint_flag_clear(uint32_t exint_line)
{
  EXINT->intsts = exint_line;
}
```

4.5 EXINT 标志位获取

flag_status exint_flag_get(uint32_t exint_line); 函数可以获取指定EXINT_LINE_x的中断标志位。

```
flag_status exint_flag_get(uint32_t exint_line)
{
  flag_status status = RESET;
  uint32_t exint_flag =0;
  exint_flag = EXINT->intsts & exint_line;
  if((exint_flag != (uint16_t)RESET))
  {
    status = SET;
  }
  else
  {
    status = RESET;
  }
  return status;
}
```

4.6 EXINT 软件触发

void exint_software_interrupt_event_generate(uint32_t exint_line);函数可以在指定EXINT_LINE_x产生软件中断/事件。

```
void exint_software_interrupt_event_generate(uint32_t exint_line)
{
  EXINT->swtrg |= exint_line;
}
```

4.7 EXINT 中断使能

void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state); 函数可以使能/禁用指定EXINT_LINE_x中断。

```
void exint_interrupt_enable(uint32_t exint_line, confirm_state new_state)
{
  if(new_state == TRUE)
  {
    EXINT->inten |= exint_line;
  }
  else
  {
    EXINT->inten &= ~exint_line;
  }
}
```

4.8 EXINT 事件使能

void exint_event_enable(uint32_t exint_line, confirm_state new_state); 函数可以使能/禁用指定EXINT_LINE_x事件。

```
void exint_event_enable(uint32_t exint_line, confirm_state new_state)
{
  if(new_state == TRUE)
  {
    EXINT->evten |= exint_line;
  }
  else
  {
    EXINT->evten &= ~exint_line;
  }
}
```

5 EXINT 中断向量

在使用EXINT中断时，除了配置时钟、GPIO引脚外，还需配置nvic_irq_enable();函数来使能中断，而EXINT中断使能又和AT32的中断向量表有关。

后文以AT32F403A/407为例进行说明，其他型号的AT32类似，不再赘述。下表是截取自技术手册RM的AT32F403A/407/407A的中断向量表。

表 3. AT32F403A/407/407A 的向量表

位置	优先级	优先级类型	名称	说明	地址
1	8	可设置	PVM	连到 EXINT 的电源电压检测（PVM）中断	0x0000_0044
6	13	可设置	EXINT0	EXINT 线 0 中断	0x0000_0058
7	14	可设置	EXINT1	EXINT 线 1 中断	0x0000_005C
8	15	可设置	EXINT2	EXINT 线 2 中断	0x0000_0060
9	16	可设置	EXINT3	EXINT 线 3 中断	0x0000_0064
10	17	可设置	EXINT4	EXINT 线 4 中断	0x0000_0068
23	30	可设置	EXINT9_5	EXINT 线[9：5]中断	0x0000_009C
40	47	可设置	EXINT15_10	EXINT 线[15：10]中断	0x0000_00E0
41	48	可设置	RTCAlarm	连到 EXINT 的 RTC 闹钟中断	0x0000_00E4
42	49	可设置	USBFS_WAKEUP	连到 EXINT 的 USBFS 唤醒中断	0x0000_00E8
80	87	可设置	EMAC_WKUP	连接 EXINT 的以太网唤醒中断	0x0000_0180

由上表可知，PVM输出（EXINT_LINE16）/ RTC闹钟（EXINT_LINE17）/ USB唤醒（EXINT_LINE18）/以太网唤醒（EXINT_LINE19，仅407）以及 EXINT0-4 拥有独立的中断向量号，而EXINT9_5和EXINT15_10共用一个中断向量号。

使能中断需配置void nvic_irq_enable(IRQn_Type irqn, uint32_t preempt_priority, uint32_t sub_priority);函数：

```
void nvic_irq_enable(IRQn_Type irqn, uint32_t preempt_priority, uint32_t sub_priority)
{
  uint32_t temp_priority = 0;
  /* encode priority */
  temp_priority = NVIC_EncodePriority(NVIC_GetPriorityGrouping(), preempt_priority, sub_priority);
  /* set priority */
  NVIC_SetPriority(irqn, temp_priority);
  /* enable irqn */
  NVIC_EnableIRQ(irqn);
}
```

该函数有3个入口参数：preempt_priority参数为抢占优先级，sub_priority参数为子优先级，这两个参数的取值范围由Cortex-M内核配置的中断优先级分组决定。irqn参数为上表3对应的中断向量号，编写代码时可以从at32f403a_407.h获取，“IRQn_Type”对中断向量号进行了枚举（与表3一致）。

```
typedef enum IRQn
{
    PVM_IRQn = 1,           /*!< pvm through exint line detection interrupt       */
    EXINT0_IRQn = 6,        /*!< external line0 interrupt      */
    EXINT1_IRQn = 7,        /*!< external line1 interrupt      */
    EXINT2_IRQn = 8,        /*!< external line2 interrupt      */
    EXINT3_IRQn = 9,        /*!< external line3 interrupt      */
    EXINT4_IRQn = 10,       /*!< external line4 interrupt      */
    EXINT9_5_IRQn = 23,     /*!< external line[9:5] interrupts     */
    EXINT15_10_IRQn = 40,   /*!< external line[15:10] interrupts     */
    RTCAlarm_IRQn = 41,     /*!< rtc alarm through exint line interrupt     */
    USBFSWakeUp_IRQn = 42,  /*!< usb device wakeup from suspend through exint line interrupt */
    EMAC_WKUP_IRQn = 80,    /*!< emac wakeup interrupt       */
} IRQn_Type;
```

由于在启动文件startup_at32f403a_407.s中，对EXINT的_IRQHandler进行了弱定义[WEAK]。用户在配置完void nvic_irq_enable();函数后，在工程的任意位置编写同名的中断处理函数即可（建议统一置于at32f403a_407_int.c方便代码管理和移植）。详细使用方法请参考后文案例分析章节。

```
Default_Handler PROC
                EXPORT  PVM_IRQHandler                  [WEAK]
                EXPORT  EXINT0_IRQHandler               [WEAK]
                EXPORT  EXINT1_IRQHandler               [WEAK]
                EXPORT  EXINT2_IRQHandler               [WEAK]
                EXPORT  EXINT3_IRQHandler               [WEAK]
                EXPORT  EXINT4_IRQHandler               [WEAK]
                EXPORT  EXINT9_5_IRQHandler             [WEAK]
                EXPORT  EXINT15_10_IRQHandler           [WEAK]
                EXPORT  RTCAlarm_IRQHandler             [WEAK]
                EXPORT  USBFSWakeUp_IRQHandler          [WEAK]
                EXPORT  EMAC_WKUP_IRQHandler            [WEAK]
```

6 EXINT 案例

注：所有 project 都是基于 keil 5 而建立，若用户需要在其他编译环境上使用，请参考 AT32xxx_Firmware_Library_V2.x.x\project\at_start_xxx\templates 中各种编译环境（例如 IAR6/7/8,keil 4/5）进行简单修改即可。

6.1 案例 1--外部中断配置

6.1.1 功能简介

通过配置 EXINT0（PA0）的每个上升沿产生外部中断，在中断处理函数中翻转 LED2/3/4。

6.1.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\exint\exint_config\mdk_v5

6.1.3 软件设计

1) 配置流程

- 配置系统时钟；
- 调用 AT32 板级支持包初始化按键和 LED；
- 配置 EXINT0（PA0）。
- 中断处理函数

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  system_clock_config();                         /* 系统时钟配置，默认 240MHz */
  at32_board_init();                             /* 初始化延时函数和 LED */
  at32_led_on(LED2);                             /* 点亮 LED2/3/4 */
  at32_led_on(LED3);
  at32_led_on(LED4);
  exint_line0_config();                          /* 配置外部中断 0 */
  while(1)
  {
  }
}
```

- PA0 的 GPIO 配置代码描述

```
#define USER_BUTTON_PIN          GPIO_PINS_0
#define USER_BUTTON_PORT         GPIOA
#define USER_BUTTON_CRM_CLK      CRM_GPIOA_PERIPH_CLOCK
void at32_button_init(void)
```

```
{
  gpio_init_type gpio_init_struct;
  /* 使能按键时钟（GPIOA）*/
  crm_periph_clock_enable(USER_BUTTON_CRM_CLK, TRUE);
  /* 配置 PA0 为下拉输入 */
  gpio_default_para_init(&gpio_init_struct);
  gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
  gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
  gpio_init_struct.gpio_mode = GPIO_MODE_INPUT;
  gpio_init_struct.gpio_pins = USER_BUTTON_PIN;
  gpio_init_struct.gpio_pull = GPIO_PULL_DOWN;
  gpio_init(USER_BUTTON_PORT, &gpio_init_struct);
}
```

■ exint_line0_config();外部中断 0 配置函数代码描述

```
void exint_line0_config(void)
{
  exint_init_type exint_init_struct;
  /* 使能 GPIOA 时钟 */
// crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  crm_periph_clock_enable(CRM_GPIOA_PERIPH_CLOCK, TRUE);
  /* 选择 PA0 作为 EXINT0 的输出 */
  gpio_exint_line_config(GPIO_PORT_SOURCE_GPIOA, GPIO_PINS_SOURCE0);
  /* 配置 EXINT_LINE_0 的上升沿产生外部中断 */
  exint_default_para_init(&exint_init_struct);
  exint_init_struct.line_enable = TRUE;
  exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_select = EXINT_LINE_0;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);
  /* 将中断优先级分组设置为 GROUP4（16 组抢占优先级（0-15），1 组子优先级） */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  /* 使能 EXINT0，将其优先级分组设定为抢占优先级 1，子优先级 0 */
  nvic_irq_enable(EXINT0_IRQn, 1, 0);
}
```

■ EXINT0_IRQHandler();外部中断 0 中断处理函数代码描述

```
void EXINT0_IRQHandler(void)
{
/* 确认 EXINT_LINE_0 状态位置起（PA0 上升沿，USER 键摁下） */
```

```
    if(exint_flag_get(EXINT_LINE_0) != RESET)
    {
      at32_led_toggle(LED2);                          /* 翻转 LED2/3/4 */
      at32_led_toggle(LED3);
      at32_led_toggle(LED4);
      exint_flag_clear(EXINT_LINE_0);                 /* 清除标志位 */
    }
  }
```

- at32_board_init();板级支持包初始化代码描述

```
  void at32_board_init()
  {
    delay_init();                               /* 初始化延时函数 */
    at32_led_init(LED2);                        /* 初始化 AT-START 板的 LED */
    at32_led_init(LED3);
    at32_led_init(LED4);
    at32_led_off(LED2);                         /* 关闭 AT-START 板的 LED */
    at32_led_off(LED3);
    at32_led_off(LED4);
    at32_button_init();                         /* 初始化 AT-START 板的 USER 按键（PA0）
  */
  }
```

- at32_led_init();LED 初始化代码描述

```
  void at32_led_init(led_type led)
  {
    gpio_init_type gpio_init_struct;
    crm_periph_clock_enable(led_gpio_crm_clk[led], TRUE); /* 使能 LED 使用的 GPIOx 时钟 */
    gpio_default_para_init(&gpio_init_struct);  /* 设置 GPIO 默认参数 */
    /* 配置 LED 的 GPIO 脚为推挽输出 */
    gpio_init_struct.gpio_drive_strength = GPIO_DRIVE_STRENGTH_STRONGER;
    gpio_init_struct.gpio_out_type  = GPIO_OUTPUT_PUSH_PULL;
    gpio_init_struct.gpio_mode = GPIO_MODE_OUTPUT;
    gpio_init_struct.gpio_pins = led_gpio_pin[led];
    gpio_init_struct.gpio_pull = GPIO_PULL_NONE;
    gpio_init(led_gpio_port[led], &gpio_init_struct);
  }
  /****************** define led @file at32f403a_407_board.h******************/
```

```
typedef enum
{
  LED2 = 0,
  LED3 = 1,
  LED4 = 2
} led_type;
#define LED_NUM 3
#if defined (AT_START_F403A_V1)  || defined (AT_START_F407_V1)
#define LED2_PIN GPIO_PINS_13
#define LED2_GPIO GPIOD
#define LED2_GPIO_CRM_CLK CRM_GPIOD_PERIPH_CLOCK

#define LED3_PIN GPIO_PINS_14
#define LED3_GPIO GPIOD
#define LED3_GPIO_CRM_CLK CRM_GPIOD_PERIPH_CLOCK
#define LED4_PIN GPIO_PINS_15
#define LED4_GPIO GPIOD
#define LED4_GPIO_CRM_CLK CRM_GPIOD_PERIPH_CLOCK
#endif
/* at-start led resouce array */
gpio_type *led_gpio_port[LED_NUM] = {LED2_GPIO, LED3_GPIO, LED4_GPIO};
uint16_t led_gpio_pin[LED_NUM] = {LED2_PIN, LED3_PIN, LED4_PIN};
crm_periph_clock_type led_gpio_crm_clk[LED_NUM] = {LED2_GPIO_CRM_CLK, LED3_GPIO_CRM_
CLK, LED4_GPIO_CRM_CLK};
```

- at32_led_toggle();LED 翻转代码描述

```
void at32_led_toggle(led_type led)
{
  if(led > (LED_NUM - 1))                                /* 判断 LED 是否有效 */
    return;
  if(led_gpio_pin[led])
    led_gpio_port[led]->odt ^= led_gpio_pin[led]; /* 通过异或配置 GPIOx_ODT 寄存器翻转 GPIO */
}
```

6.1.4 实验效果

上电运行，每次摁下 USER 键，观察到 LED2、LED3 和 LED4 翻转，符合程序设计预期。

6.2 案例 2--EXINT 软件触发

6.2.1 功能简介

通过在 TMR1 的溢出中断处理函数配置软件触发 EXINT4。在 TMR1 的溢出中断处理函数翻转 LED2，在 EXINT4 的中断处理函数翻转 LED3/4，用以指示程序执行状态。

6.2.2 资源准备

1) 硬件环境：

AT32F403A 的 AT-START BOARD

2) 软件环境：

\project\at_start_f403a\examples\exint\exint_software_trigger\mdk_v5

6.2.3 软件设计

1) 配置流程

- 配置系统时钟；
- 调用 AT32 板级支持包初始化 LED；
- 配置 TMR1 及 EXINT4；
- TMR1 与 EXINT4 中断处理函数。

2) 代码介绍

- main 函数代码描述

```
int main(void)
{
  /* 将中断优先级分组设置为 GROUP4（16 组抢占优先级（0-15），1 组子优先级） */
  nvic_priority_group_config(NVIC_PRIORITY_GROUP_4);
  system_clock_config();                          /* 系统时钟配置，默认 240MHz */
  at32_board_init();                              /* 初始化延时函数和 LED */
  at32_led_on(LED2);                              /* 点亮 LED2/3/4 */
  at32_led_on(LED3);
  at32_led_on(LED4);
  tmr1_config();                                  /* 配置 TMR1 */
  exint_line4_config();                           /* 配置外部中断 4 */
  while(1)
  {
  }
}
```

- tmr1_config();函数代码描述

```
static void tmr1_config(void)
{
  crm_clocks_freq_type crm_clocks_freq_struct = {0};
  /* 获取系统时钟频率 */
```

```
  crm_clocks_freq_get(&crm_clocks_freq_struct);
  /* 开启 TMR1 时钟 */
  crm_periph_clock_enable(CRM_TMR1_PERIPH_CLOCK, TRUE);
  /* 配置 TMR1 溢出中断时间为 1s (systemclock / (system_core_clock/10000)) / 10000 = 1Hz(1s) */
  tmr_base_init(TMR1, 10000-1, system_core_clock/10000-1);
  /* 配置计数方向为向上计数 */
  tmr_cnt_dir_set(TMR1, TMR_COUNT_UP);
  /* 配置时钟除频参数为不除频（该时钟用于滤波及死区生成） */
  tmr_clock_source_div_set(TMR1, TMR_CLOCK_DIV1);
  /* 使能 TMR1 的溢出中断 */
  tmr_interrupt_enable(TMR1, TMR_OVF_INT, TRUE);
// tmr_flag_clear(TMR1,TMR_OVF_FLAG);
  /* 配置 TMR1_OVF_TMR10 中断，将其优先级分组设定为抢占优先级 0，子优先级 0 */
  nvic_irq_enable(TMR1_OVF_TMR10_IRQn, 0, 0);
}
```

- exint_line4_config();外部中断 4 配置函数代码描述

```
void exint_line4_config(void)
{
  exint_init_type exint_init_struct;
// crm_periph_clock_enable(CRM_IOMUX_PERIPH_CLOCK, TRUE);
  exint_default_para_init(&exint_init_struct);
  /* 配置 EXINT_LINE_4 的上升沿产生外部中断 */
  exint_init_struct.line_enable = TRUE;
  exint_init_struct.line_mode = EXINT_LINE_INTERRUPUT;
  exint_init_struct.line_select = EXINT_LINE_4;
  exint_init_struct.line_polarity = EXINT_TRIGGER_RISING_EDGE;
  exint_init(&exint_init_struct);
  /* 使能 EXINT4，将其优先级分组设定为抢占优先级 1，子优先级 0 */
  nvic_irq_enable(EXINT4_IRQn, 1, 0);
}
```

- TMR1_OVF_TMR10_IRQHandler();TMR1 溢出中断处理函数代码描述

```
void TMR1_OVF_TMR10_IRQHandler(void)
{
  /* 确认 TMR1 的 TMR_OVF_FLAG 状态位置起（TMR1 溢出事件发生） */
  if(tmr_flag_get(TMR1,TMR_OVF_FLAG) != RESET)
  {
```

```
        at32_led_toggle(LED2);                                  /* 翻转 LED2 */
        exint_software_interrupt_event_generate(EXINT_LINE_4);  /* 软件触发 EXINT4 */
        tmr_flag_clear(TMR1,TMR_OVF_FLAG);                      /* 清除标志位 */
    }
}
```

- EXINT4_IRQHandler();外部中断 4 中断处理函数代码描述

```
void EXINT4_IRQHandler(void)
{
  /* 确认 EXINT_LINE_4 状态位置起（由 TMR1 溢出中断处理函数配置软件触发） */
  if(exint_flag_get(EXINT_LINE_4) != RESET)
  {
    at32_led_toggle(LED3);                 /* 翻转 LED3/4 */
    at32_led_toggle(LED4);
    exint_flag_clear(EXINT_LINE_4);        /* 清除标志位 */
  }
}
```

6.2.4 实验效果

上电运行，观察到 LED2、LED3 和 LED4 每隔 1s 翻转一次，符合程序设计预期。

复位后，LED3/LED4 点亮，LED2 熄灭，这是因为 tmr_base_init();函数有置位软件事件寄存器（TMRx_SWEVT）的 bit0 软件触发溢出事件 OVFSWTR 来立即更新 TMR 的初始化参数。在 nvic_irq_enable();使能后会立即进入一次 TMR1_OVF_TMR10_IRQHandler();来翻转 LED2。

这样设计程序是为了方便用户区分进入 TMR1 和 EXINT4 中断的先后顺序，如果用户想要 LED2 和 LED3/LED4 状态同步，在 tmr1_config();函数内的 tmr_base_init();函数后，nvic_irq_enable();函数前加入清除标志位的代码 tmr_flag_clear(TMR1,TMR_OVF_FLAG);即可。

更多关于 TMR 特性的介绍，用户可以参考应用笔记《AN0085_AT32_MCU_定时器入门指南》，这些资料可以从雅特力科技官方网站获取。

7 文档版本历史

表 4. 文档版本历史

日期	版本	变更
2022.04.15	2.0.0	最初版本

重要通知-请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力产品并非设计或专门用于下列用途的产品：(A)对安全性有特别要求的应用，如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B)航空应用；(C)汽车应用或汽车环境；(D)航天应用或航天环境，且/或(E)武器。因雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，风险由购买者单独承担，并且独力负责在此类相关使用中满足所有法律和法规要求。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。

©2022 雅特力科技(重庆)有限公司保留所有权利