HPM6700/6400 系列微控制器

通用输入输出控制器

GPIO 使用介绍

目录

版本:

日期	版本号	说明
2022-8-9	1.0	初版

1 简介

HPM6700/6400 系列 MCU 是来自上海先楫半导体科技有限公司的高性能实时 RISC-V 微控制器，为工业自动化及边缘计算应用提供了极大的算力、高效的控制能力。上海先楫半导体目前已经发布了如 HPM6700/6400、HPM6300 等多个系列的高性能微控制器产品。

在 HPM6700/6400 系列微控制器上有非常丰富的输入输出模块，通过 GPIO 管理器来管理各 GPIO 控制器的 IO 控制权限，其中的快速 GPIO 控制器可以让其控制的 IO 最大翻转率可达 CPU 主频的一半。还有特殊的两组特殊 IO:PY,PZ，作为电源管理域和电池备份域的外设，PY 和 PZ 由 PIOC 和 BIOC 控制时，可以在系统电源域和电源管理域掉电时保持工作。

本文提供了与 HPM6700/6400 系列微控制器 GPIO 模块的功能描述，以及它们各自的特点，并提供了如何使用这些 GPIO 的建议。

2 GPIO 模块特点简介

本节简单介绍通用输入输出控制器 GPIO 的主要功能和特点。GPIO 的主要特点有：

- 丰富的 GPIO 功能复用引脚
- 配置 IO 作为输出或者输入
- GPIOM 模块配置决定具体哪个 GPIO 控制器来控制通用 IO
- 快速 GPIO 控制器控制 IO 时，IO 最大翻转率可达 CPU 主频的一半
- PY 的 IO 由 VPMC 供电，并由电源管理域 IO 控制器 PIOC 控制
- PZ 的 IO 由 VBAT 供电，并由电池备份域 IO 控制器 BIOC 控制

HPM6700/6400 系列的 MCU 输入输出提供 PA~PZ 共 8 组最多 195 个 GPIO 功能复用引脚，每个 GPIO 都可以由 2 个 GPIO 控制器和 2 个快速 GPIO 控制器控制，由 GPIO 管理器 GPIOM 指定。提到的 2 个快速

GPIO 控制器 FGPIO，作为处理器私有的 IO 快速访问接口，处理器因此可以零等待周期来访问 FGPIO 控制器。有两组特殊 IO：PY,PZ, 作为电源管理域和电池管理域的专属 GPIO 控制器和 IO 配置模块，可以在低功耗模式下保持。并且 BGPIO 或是 PGPIO 中断能在系统电源域和电源管理域掉电时把系统唤醒。

3 GPIO 控制器介绍

本节将介绍 HPM6700/6400 系列的通用输入输出控制器（General Purpose Input Output），GPIO 控制器包括： 2 个 GPIO 控制器（GPIO0,GPIO1）， 2 个快速 GPIO 控制器（FGPIO0， FGPIO1），电源管理域 GPIO 控制器（PGPIO）和电池备份域 GPIO 控制器 (BGPIO) 。

3.1 通用 GPIO 控制器控制

GPIO 控制器与快速 GPIO 控制器功能基本相同，可以按照 IO 端口 Port 读取输入，配置 IO 作为输入或者输出，设置 IO 输出，或者同时把一个或者多个 IO 输出设置高，设置低或者翻转。GPIOx 和 FGPIOx 可以控制通用 IO（PA， PB， PC， PD， PE， PF）。

GPIO 模块配备了简单易用的寄存器，易于用户开发使用，本节将对这些寄存器的使用做一个介绍。

GPIO 控制器支持 OE 寄存器，作为 GPIO 的方向控制寄存器，每一个 IO 都有对应的 DIRECTION 控制位。用户把该位置 1 就可以把对应的 IO 配置为 GPIO 输出，如果置 0，那么此 IO 就是 GPIO 输入。

如果想配置 GPIO 的输出，可以使用 GPIO 寄存器中的 DO 寄存器来配置。与之对应的，可以用过 DI 寄存器来读取 IO 的电平状态，值得一提的是，无论 GPIO 配置是输入还是输出，或者对应 IO 控制器 IOC 是否将 IO 功能映射为 GPIO，都可以通过 DI 寄存器来读取到 IO 的状态。

配合这几种寄存器的使用，GPIO 的输入输出还支持如下几种原子化操作寄存器。

- SET 是输出高寄存器，例如在 DO【SET】中有 31-0 的位域，每一位代表一个引脚，写 1 则置高，写 0 没有影响。
- CLEAR 是输出低寄存器，例如在 DO【CLEAR】中有 31-0 的位域，每一位代表一个引脚，写 1 则置低，写 0 没有影响。
- TOGGLE 是翻转寄存器，例如在 DO【TOGGLE】中有 31-0 的位域，每一位代表一个引脚，写 1 则会把对应 IO 输出翻转，写 0 没有影响。

使用这几种寄存器就可以完成高低电平的配置，置 0 引脚输出低电平，置 1 引脚输出高电平。最基本的点灯就可以通过 Gpio_wirte_pin 写入 0 和 1 配置高低电平，完成 LED 灯的点亮。本文将使用 HPM6750EVKMINI 开发板举例。例如图 1 中的红色 LED 的点灯的代码。

```
gpio_write_pin(BOARD_R_GPIO_CTRL, BOARD_R_GPIO_INDEX, BOARD_R_GPIO_PIN, 0);
    board_delay_ms(LED_FLASH_PERIOD_IN_MS);
gpio_write_pin(BOARD_R_GPIO_CTRL, BOARD_R_GPIO_INDEX, BOARD_R_GPIO_PIN, 1);
    board_delay_ms(LED_FLASH_PERIOD_IN_MS);
```

图表 1. 红色小灯闪烁

可以看到如图 2 所示，在 Gpio_wirte_pin 的声明中便是使用 GPIO 寄存器中的 DO 寄存器来配置，配合 SET 和 CLEAR 分别置 1 完成高低电平的配置,从而可以使用此函数。

```
static inline void gpio_write_pin(GPIO_Type *ptr, uint32_t port, uint8_t pin, uint8_t high)

    if (high) {
        ptr->DO[port].SET = 1 << pin;
    } else {
        ptr->DO[port].CLEAR = 1 << pin;
```

图表 2. drv.h 中的定义

如图 3 所示，在 HPM 官方公布的 SDK 包中就有 GPIO 的一个例程，控制 LED 灯的多次闪烁，其中闪灯的部分代码循环中就使用了 TOGGLE 寄存器来控

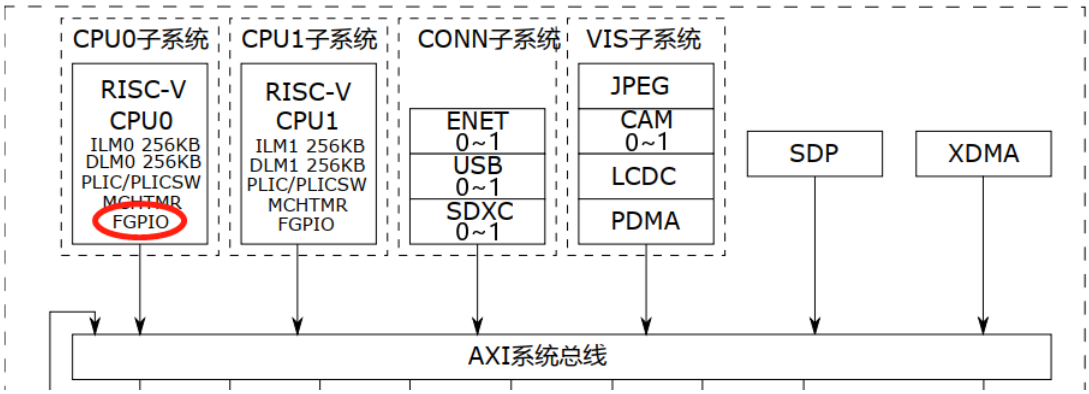
制 IO 输出翻转，控制 GPIO 引脚输出电平变化，方便简洁。

```
for (uint32_t i = 0; i < GPIO_TOGGLE_COUNT; i++)
        gpio_toggle_pin(BOARD_LED_GPIO_CTRL, BOARD_LED_GPIO_INDEX,
                        BOARD_LED_GPIO_PIN);
        board_delay_ms(500);
        gpio_toggle_pin(BOARD_LED_GPIO_CTRL, BOARD_LED_GPIO_INDEX,
                        BOARD_LED_GPIO_PIN);
        board_delay_ms(500);
        printf("toggling led %u/%u times\n", i + 1, GPIO_TOGGLE_COUNT);
```

图表 3. GPIO 引脚输出电平变化

3.2 快速 GPIO 控制器

快速 GPIO 控制器属于处理器的私有外设，如图 4 所示，处理器因此可以零等待周期来访问 FGPIO 控制器。当 IO 由 FGPIO 控制时，IO 最大翻转率可达 CPU 主频的一半，以 HPM6750 CPU 运行在 816MHz 为例，IO 翻转频率可 408MHz。

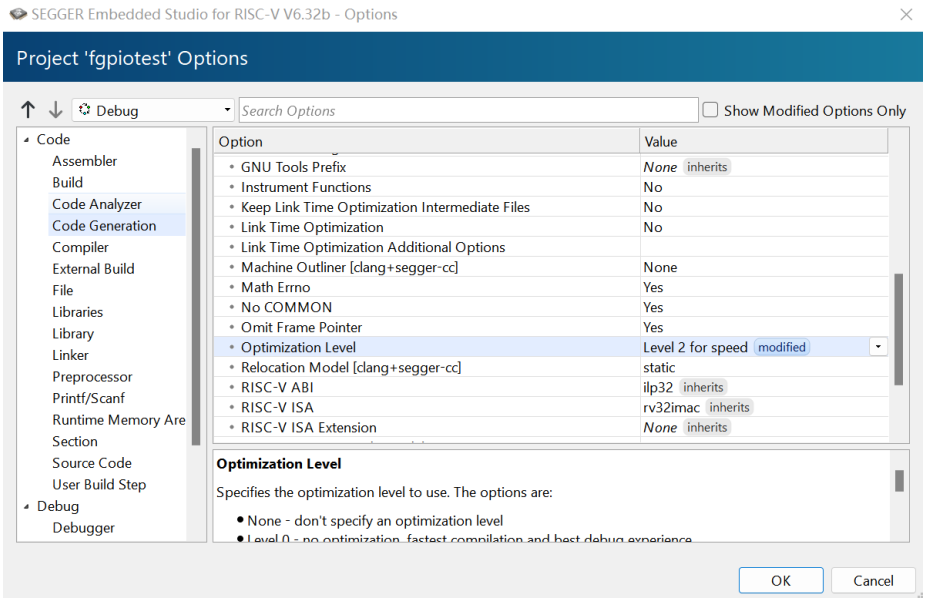


图表 4. 系统架构框图

通过在程序中选用 FGPIO0 或者 FGPIO1 这两个控制器来控制 IO，将 IO 配置为输出后，可以使用上文提到的 DO【TOGGLE】寄存器，令这个 IO 循环反复做一个翻转，部分代码如图 5 所示：将 PD16 配置完成。需要注意的是，在测试波形之前，需要在 Project 的选项中打开优化设置，如图 6 所示：将 Optimization Level 设定为 Level 2 for speed。

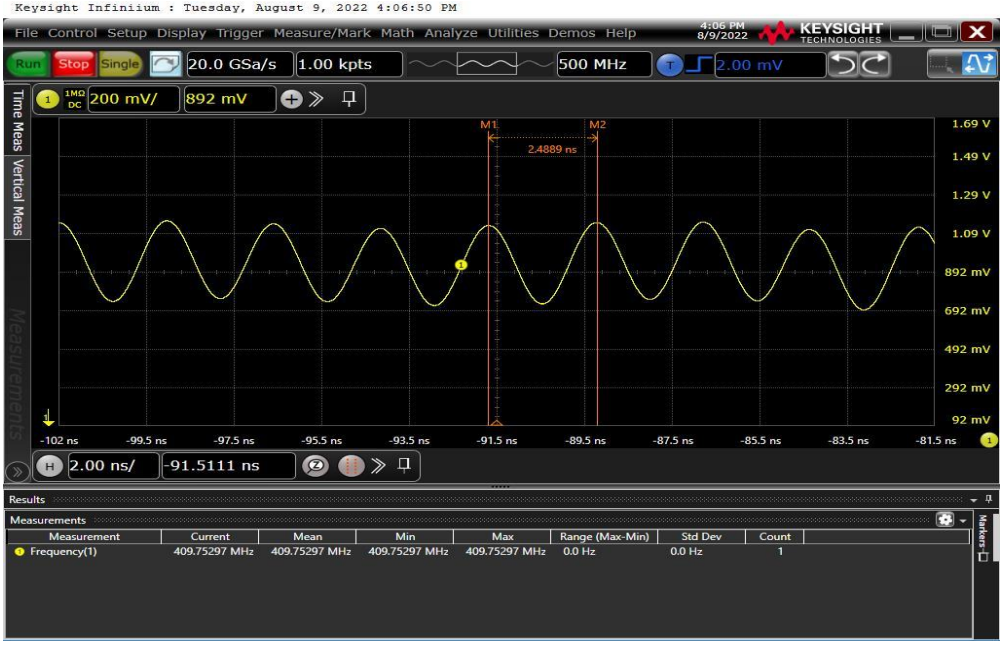
```
HPM_IOC->PAD[IOC_PAD_PD16].FUNC_CTL = IOC_PD16_FUNC_CTL_GPIO_D_16;
gpiom_set_pin_controller(HPM_GPIOM, GPIO_DI_GPIOD, 16, gpiom_core0_fast);
gpio_set_pin_output(HPM_FGPIO, 3, 16);
 while (1) {
    gpio_toggle_pin(HPM_FGPIO, GPIO_DO_GPIOD, 16);
 }
 return 0;
```

图表 5. FGPIO 控制 IO 翻转代码



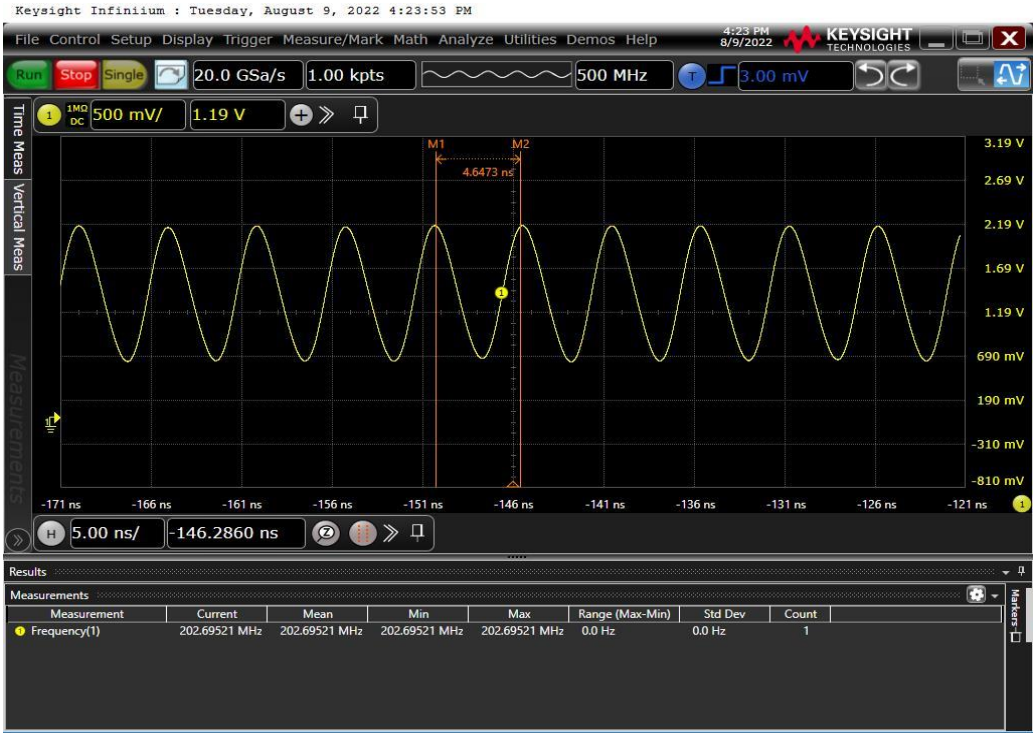
图表 6. 开启优化

设定完成后，就可以使用示波器连接到此 IO 后，得到的波形如图 7.1 所示，测量峰-峰值，此时的频率为 409.7MHz, 的确达到了 CPU 主频的一半。

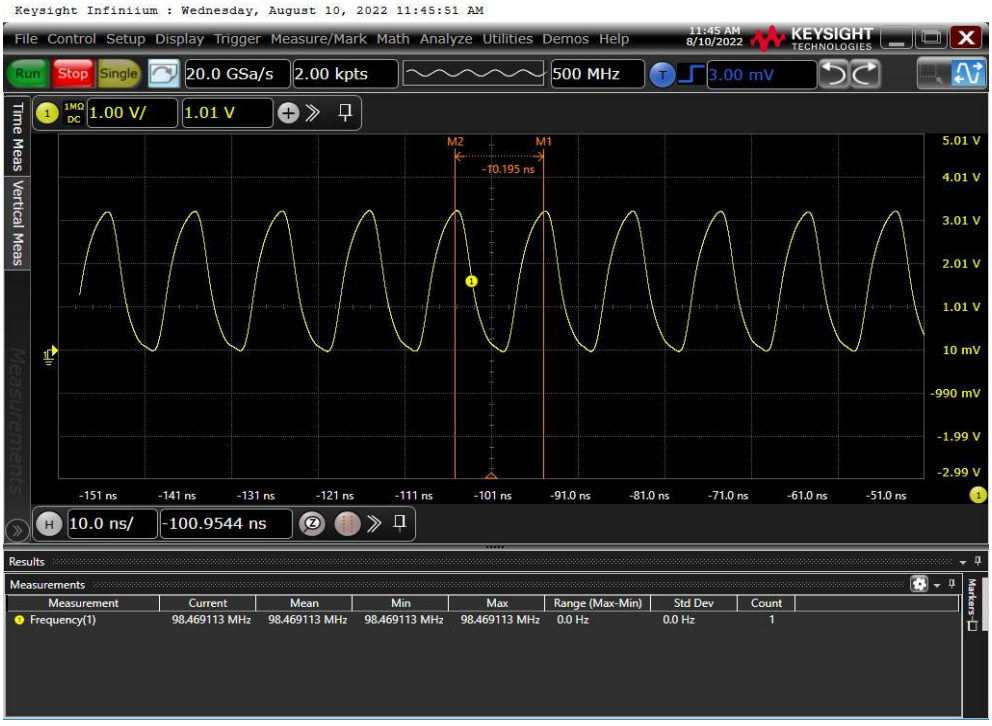


图表 7. 1 FGPIO 控制 IO 的翻转波形

由于 FGPIO 控制 IO 的情况下，IO 的翻转频率为主频的一半，那么修改当前的主频频率，还可以测出其他几种频率的翻转波形，如图 7.2 与 7.3 所示，分别是 200Mhz 和 100Mhz 频率的翻转波形,它们对应的主频分别是 400Mhz 与 200Mhz。

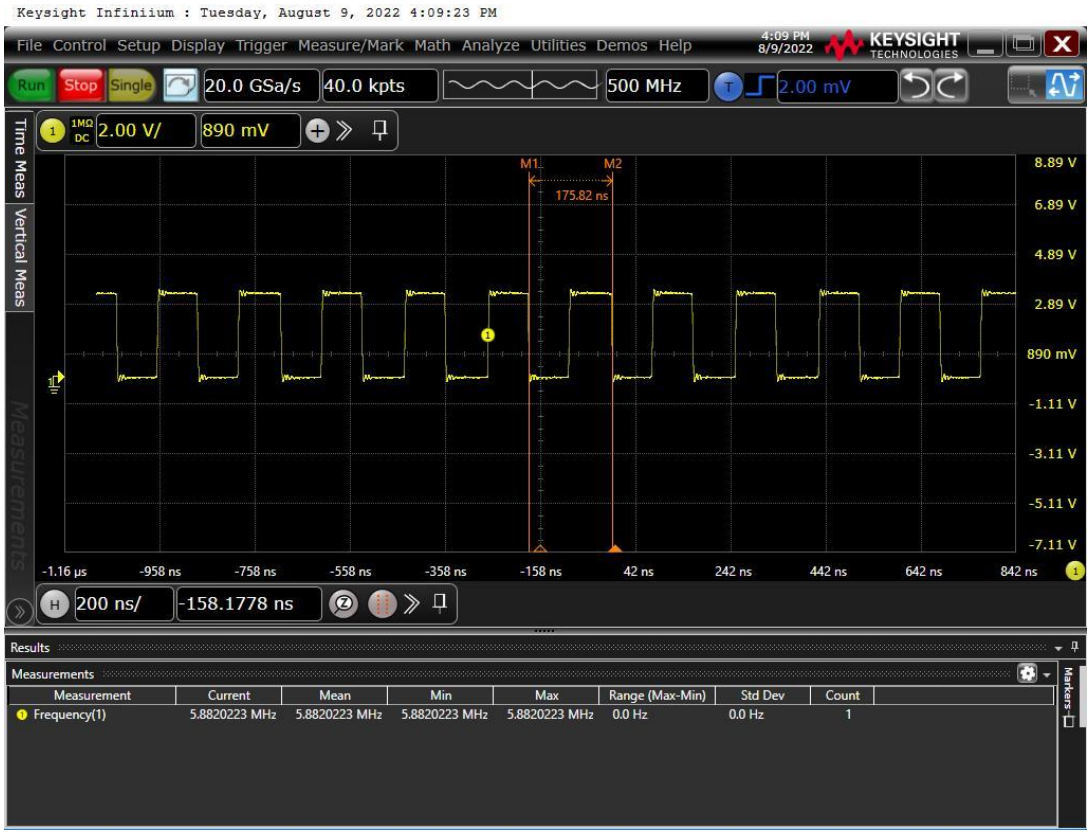


图表 7. 2 FGPIO 控制 IO 下 200mhz 波形



图表 7. 3 FGPIO 控制 IO 下 100mhz 波形

HPM6700/6400 系列 MCU 中的普通 GPIO 控制器控制 IO 时，也可以有 5.8Mhz 左右的翻转频率，图 7.4 为 GPIO0 控制 IO 时，用示波器抓取的波形。



图表 7. 4 GPIO 控制 IO 下的翻转波形

3.3 通用 GPIO 控制器中断

本小节将介绍 GPIO 控制器触发中断的功能，GPIO 控制器支持配置和生成 GPIO 中断，需要注意的是，快速 FGPIO 不支持生成中断，GPIO 有关中断的几种寄存器如下：

- IE 寄存器打开 GPIO 中断，IE 寄存器内的对应位置 1 就可以使能对应 IO 的中断。
- IF 寄存器可以用来查询中断的状态，对应的标志位如果是 1，代表产生了中断；如果是 0，则没有产生中断。
- PL 寄存器可以用来指定中断的极性，对应位置 1，表示中断由下降沿或者低电平触发；对应位置 0，表示中断由上升沿或高电平触发。

- TP 寄存器可以用来指定中断的类型，对应位置 1，表示中断由边沿触发；对应位置 0，表示中断由电平触发。
- AS 寄存器可以让 GPIO 以异步方式产生中断清零，对应位置 0，代表使用系统时钟产生中断；对应位置 1，代表利用组合逻辑产生中断。

图 8 是一段实现输入引脚触发中断功能的部分代码，在配置之后可以通过按下 GPIO 按键让 LED 灯亮起或是熄灭。

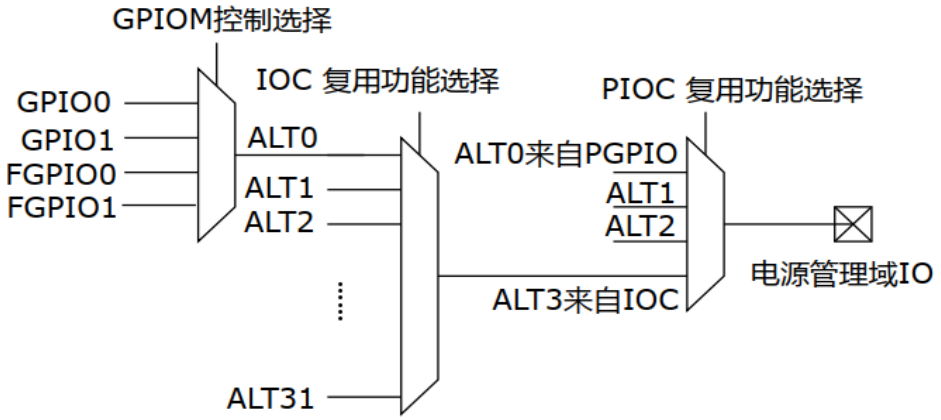
```
gpio_set_pin_input(BOARD_APP_GPIO_CTRL, BOARD_APP_GPIO_INDEX, BOARD_APP_GPIO_PIN);

    trigger = gpio_interrupt_trigger_edge_falling;

    gpio_config_pin_interrupt(BOARD_APP_GPIO_CTRL, BOARD_APP_GPIO_INDEX,
                              BOARD_APP_GPIO_PIN, trigger);
    gpio_enable_pin_interrupt(BOARD_APP_GPIO_CTRL, BOARD_APP_GPIO_INDEX,
                              BOARD_APP_GPIO_PIN);
```

图表 8. 中断的配置例子

在 set_pin 声明中，通过上文所提到的 OE 寄存器，置 0 将 IO 配置为 GPIO 输入，接着在 config 的声明中，使用 PL 和 TP 的寄存器分别声明了电平以及边沿触发中断的情况。此段代码中将 trigger 编写为沿下降沿触发，随后通过 enable 声明中的 IE 寄存器打开 IO 中断。

3.4 PGPIO 与 BGPIO

在 HPM6700/6400 系列中，有两组特殊 IO：PY,PZ。它们分别是电源管理域 IO（PY）以及电池备份域 IO（PZ）。PY 的 IO 由 VPMC 供电，并由电源管理域 IO 控制器 PIOC 控制。PZ 的 IO 由 VBAT 供电，并由电池备份域 IO 控制器 BIOC 控制。PIOC 和 BIOC 可以把电源管理域 IO（PY）和电池备份域 IO（PZ）中的一个或者多个 IO 映射到系统电源域。之后，这些 IO 就可以由 GPIOx 或 FGPIOx 控制。电源管理域的控制选择如图 9 所示。



图表 9. 电源管理域 IO GPIO 控制选择

从图中不难看出，相比于普通 IO 的配置，不仅仅要配置 IOC，如果想让 PY，也就是电源管理域的 IO 分配到系统电池域外设，还有配置 PIOC。同样对于 PZ 这个电池备份域的 IO 也是一样的，还需要配置 BIOC。图 10 是 HPM6700 微控制器中的一个例子，此段代码将 PY07，PY06 配置为 UART0_RX 和 UART0_TX，将 PZ08，PZ09 配置为 UART13_RX 和 UART13_TX。

```
if (ptr == HPM_UART0) {
        HPM_IOC->PAD[IOC_PAD_PY07].FUNC_CTL = IOC_PY07_FUNC_CTL_UART0_RXD;
        HPM_IOC->PAD[IOC_PAD_PY06].FUNC_CTL = IOC_PY06_FUNC_CTL_UART0_TXD;
        /* PY port IO needs to configure PIOC as well */
        HPM_PIOC->PAD[IOC_PAD_PY07].FUNC_CTL = IOC_PY06_FUNC_CTL_SOC_PY_06;
        HPM_PIOC->PAD[IOC_PAD_PY06].FUNC_CTL = IOC_PY07_FUNC_CTL_SOC_PY_07;
    } else if (ptr == HPM_UART6) {
        HPM_IOC->PAD[IOC_PAD_PE27].FUNC_CTL = IOC_PE27_FUNC_CTL_UART6_RXD;
        HPM_IOC->PAD[IOC_PAD_PE28].FUNC_CTL = IOC_PE28_FUNC_CTL_UART6_TXD;
    } else if (ptr == HPM_UART7) {
        HPM_IOC->PAD[IOC_PAD_PC02].FUNC_CTL = IOC_PC02_FUNC_CTL_UART7_RXD;
        HPM_IOC->PAD[IOC_PAD_PC03].FUNC_CTL = IOC_PC03_FUNC_CTL_UART7_TXD;
    } else if (ptr == HPM_UART13) {
        HPM_IOC->PAD[IOC_PAD_PZ08].FUNC_CTL = IOC_PZ08_FUNC_CTL_UART13_RXD;
        HPM_IOC->PAD[IOC_PAD_PZ09].FUNC_CTL = IOC_PZ09_FUNC_CTL_UART13_TXD;
        /* PZ port IO needs to configure BIOC as well */
        HPM_BIOC->PAD[IOC_PAD_PZ08].FUNC_CTL = IOC_PZ08_FUNC_CTL_SOC_PZ_08;
        HPM_BIOC->PAD[IOC_PAD_PZ09].FUNC_CTL = IOC_PZ09_FUNC_CTL_SOC_PZ_09;
```

图表 10. 配置 PY 以及 PZ 的代码例子

可以注意到，代码中还有将普通 IO：PE 和 PC 配置为 UART6，UART7 的引脚功能，与 PY 和 PZ 的区别正是少配置了 PIOC 和 BIOC，使用这组特殊的

PGPIO 以及 BGPIO 可以在系统电源域和电源管理域掉电时保持工作，它们中断能在系统电源域掉电时把系统唤醒。

4 GPIO 管理器 GPIOM 介绍

GPIO 管理器是一个能为任一 IO 指定 GPIO 配置生效的模块，管理器的主要特点如下：

- 可以为 IO 分配指定的 GPIO 控制器，作为一个管理权限，可以从两个 GPIO 控制器和 FGPIO 控制器里任意选择。
- 可以配置 IO 输入是否对特定的 GPIO 控制器可见。
- 可以锁定一个 IO 的对应寄存器

HPM6700/6400 系列可以支持多个 GPIO 控制器，目的就是为了让 GPIO 资源隔离，从而方便多核，多任务的实现。此模块寄存器为 ASSIGN【PIN】，一共有 32 位域，修改不同位域的寄存器值可以实现 GPIOM 的配置。

1-0 位域作为 SELECT 位域，可以选择这个 IO 受到哪个 GPIO 控制：置 0 为 GPIO0;置 1 为 GPIO1;置 2 为 CPU 快速 GPIO；置 3 为 CPU1 快速 GPIO。

11-8 位作为 HIDE 位域，由 4 位组成，每一位置 1，这个 IO 的输入在对应的 GPIO 控制器 DI 寄存器内不可见，那么 GPIO 控制器无法读取到这个 IO 的输入。0 位代表 GPIO0 是否可见；1 位代表 GPIO1 是否可见；2 位代表 CPU0 快速 GPIO0 是否可见；3 位代表 CPU1 快速 GPIO1 是否可见。

31 位用作 LOCK 位域，用来锁定该寄存器的设置，当此位置 1 后，寄存器字段就不可修改了，在锁定后只能通过复位来清零。

如图 11 所示，在 HPM 官方公布的 SDK 包中有一个有关 GPIOM 的例程，演示了如何使用 GPIOM 模块设置引脚的控制模块，之后用指定的控制模块来改变引脚的状态，点亮 LED 灯。

```
gpiom_set_pin_controler(BOARD_APP_GPIOM_BASE, BOARD_LED_GPIO_INDEX, BOARD_LED_GPIO_PIN, gpio_module);
    gpiom_enable_pin_visibility(BOARD_APP_GPIOM_BASE, BOARD_LED_GPIO_INDEX, BOARD_LED_GPIO_PIN, gpio_module);
    gpiom_lock_pin(BOARD_APP_GPIOM_BASE, BOARD_LED_GPIO_INDEX, BOARD_LED_GPIO_PIN);
```

图表 11. GPIOM 设置引脚例程的部分代码

这三行代码分别就是对应了 GPIOM 位域的寄存器配置：SELECT, HIDE,LOCK。与上文中的 GPIO 配置类似，在此例程中的 hpm_gpiom_drv.h 中已经通过配置 ASSIGH 寄存器的不同位域，声明了一些函数，包括 GPIO 分配，访问控制等。

在 ASSIGH 寄存器中还有 NON_SEC 位，可以按照系统安全状态进行访问权限控制。一旦置 1，寄存器只能在安全状态下访问。在 HPM6700/6400 用户手册中的系统安全相关章节，可以获取系统安全状态的详细信息。

5 总结

本文主要介绍了 HPM6700/6400 系列高性能微控制器的 GPIO 基本功能与特点。这包括控制器与快速控制器来配置输入输出，IO 高达 400Mhz 的翻转速率，特殊的两组 IO 引脚配置；可以实现 GPIO 资源隔离的管理模块 。

本文同时也分享了一些相关代码和相关寄存器的使用方式，用户可以通过查阅 HPM6700/6400 用户手册中相关寄存器的详细介绍，根据自己的实际需求来配置 GPIO。