V459 melis

GPIO 模块使用说明

1.0

2020.03.12

文档履历

版本号	日期	制/修订人	内容描述
1.0	2020.03.12		version 1.0

目录

1. 概述

1.1 编写目的

介绍 melis-RTOS 中 GPIO 驱动的的接口以及使用方法，为 GPIO 驱动的使用者提供参考

1.2 适用范围

适用于 V459 配套的 melis 系统平台

1.3 相关人员

GPIO 驱动驱动层/应用层的开发/使用/维护人员

2. 模块介绍

2.1 模块功能介绍

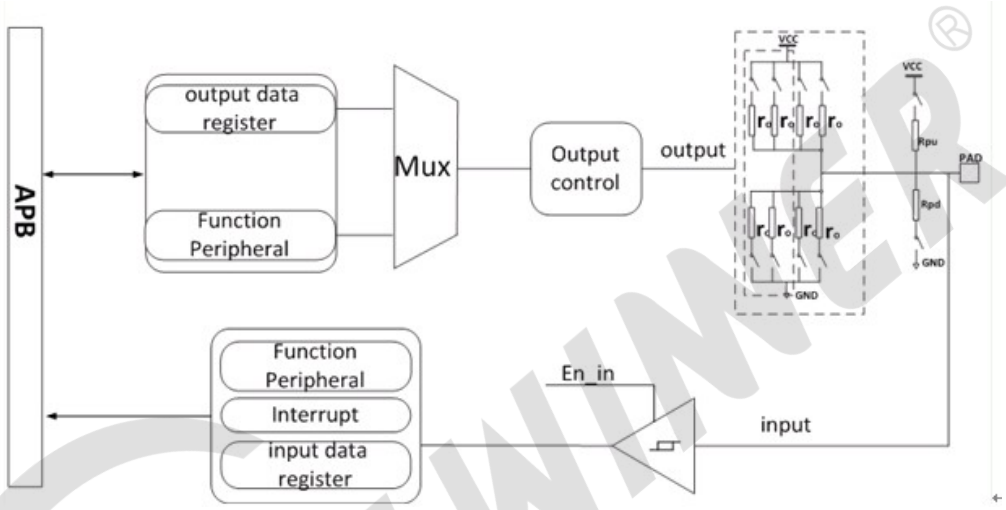

图 1: PORT 控制器

整个 GPIO 控制器由数字部分（GPIO 和外设接口）以及 IO 模拟部分（输出缓冲，双下拉，pad）组成。其中数字部分的输出可以通过 MUX 开关选择，模拟部分可以用来配置上下拉，驱动能力以及引脚输出电压等等。具体的规格如下：

- 支持 8 组 GPIO 引脚（PC，PD，PE，PF，PG, PH，PI，PL）
- 可以在软件上配置各个引脚的状态
- 每个引脚都可以触发中断
- 可以配置上拉/下拉/无上下拉三种状态
- 每个引脚都可以配置 4 种驱动能力
- 可以配置边缘中断触发
- 最高 99 个中断

2.2 模块配置介绍

GPIO 的基本配置如下所示，主要是配置 GPIO 模块的控制器地址以及每个 GPIO 的中断号，具体如下所示：

```
#define PA_BASE 0
#define PB_BASE 32
#define PC_BASE 64
#define PD_BASE 96
#define PE_BASE 128
#define PF_BASE 160
#define PG_BASE 192
#define PH_BASE 224
#define PI_BASE 256
#define PJ_BASE 288
#define PK_BASE 320
#define PL_BASE 352
#define PM_BASE 384
#define PN_BASE 416
#define PO_BASE 448
#define GPIO_MAX_BANK PL_BASE
#define SUNXI_GPIO_PBASE 0x0300B000    //CPUX域GPIO的基地址
#define SUXNI_GPIO_R_PBASE 0x07022000  //CPUS域GPIO的基地址

/* sunxi gpio name space */
#define GPIOA(n)    (PA_BASE + (n))   //定义每个引脚所在的寄存器
#define GPIOB(n)    (PB_BASE + (n))
#define GPIOC(n)    (PC_BASE + (n))
#define GPIOD(n)    (PD_BASE + (n))
#define GPIOE(n)    (PE_BASE + (n))
#define GPIOF(n)    (PF_BASE + (n))
#define GPIOG(n)    (PG_BASE + (n))
#define GPIOH(n)    (PH_BASE + (n))
#define GPIOI(n)    (PI_BASE + (n))
#define GPIOJ(n)    (PJ_BASE + (n))
#define GPIOK(n)    (PK_BASE + (n))
#define GPIOL(n)    (PL_BASE + (n))
#define GPIOM(n)    (PM_BASE + (n))
#define GPION(n)    (PN_BASE + (n))
#define GPIOO(n)    (PO_BASE + (n))

/* sunxi gpio irq*/
#define SUNXI_GIC_START 32
#define SUNXI_IRQ_GPIOC (SUNXI_GIC_START + 67)
#define SUNXI_IRQ_GPIOD (SUNXI_GIC_START + 68)
#define SUNXI_IRQ_GPIOE (SUNXI_GIC_START + 69)
#define SUNXI_IRQ_GPIOF (SUNXI_GIC_START + 70)
#define SUNXI_IRQ_GPIOG (SUNXI_GIC_START + 71)
```

```
#define SUNXI_IRQ_GPIOH (SUNXI_GIC_START + 72)
#define SUNXI_IRQ_GPIOI (SUNXI_GIC_START + 73)
#define SUNXI_IRQ_R_GPIOL (SUNXI_GIC_START + 106)
```

2.3 模块源码结构

GPIO 模块的源代码位于 ekernel 的 driver 目录下，其中 drv 层和 hal 层存放 GPIO 模块的源码，而 test 目录则存放着 GPIO 的测试用例代码。具体如下所示：

```
source/ekernel/drivers/hal/source/gpio/
|---- hal_gpio.c  //驱动层源码
source/ekernel/drivers/drv/source/gpio/
|---- drv_gpio.c  //应用层源码
source/ekernel/drivers/include/
|---- drv/sunxi_drv_gpio.h //应用层头文件
|---- hal/sunxi_hal_gpio.h //驱动层头文件
source/ekernel/drivers/test/
|---- test_gpio.c  //GPIO测试文件
```

3. 模块接口说明

3.1 数据结构

由于 GPIO 需要配置每个引脚的引脚复用功能，中断类型，驱动能力，上下拉，输出/输入数据，输入/输出方向等等，所以对 GPIO 的这些配置都封装在一个 enum 枚举结构里面，方便使用。下面是一些配置的定义。想要了解更多的可以到 sunxi_hal_gpio.h 查看

3.1.1 引脚定义 gpio_pin_t

该枚举定义了可用的每个引脚定义，在配置引脚的时候将相关参数传入则可，具体定义如下：

```
typedef enum
{
    GPIO_PC0 = GPIOC(0),
    GPIO_PC1 = GPIOC(1),
    GPIO_PC2 = GPIOC(2),
    GPIO_PC3 = GPIOC(3),
    ...
    GPIO_PL0 = GPIOL(0),
    GPIO_PL1 = GPIOL(1),
    GPIO_PL2 = GPIOL(2),
    GPIO_PL3 = GPIOL(3),
    GPIO_PL4 = GPIOL(4),
    GPIO_PL5 = GPIOL(5),
} gpio_pin_t;
```

3.1.2 引脚复用功能 gpio_muxsel_t

该枚举定义了引脚的复用功能的值，具体定义如下：

```
typedef enum
{
    GPIO_MUXSEL_IN = 0,
```

```
    GPIO_MUXSEL_OUT = 1,
    GPIO_MUXSEL_FUNCTION2 = 2,
    GPIO_MUXSEL_FUNCTION3 = 3,
    GPIO_MUXSEL_FUNCTION4 = 4,
    GPIO_MUXSEL_FUNCTION5 = 5,
    GPIO_MUXSEL_FUNCTION6 = 6,
    GPIO_MUXSEL_DISABLED = 7,
} gpio_muxsel_t;
```

3.1.3 引脚驱动能力 gpio_driving_level_t

该枚举定义了引脚的驱动能力的值，具体定义如下：

```
typedef enum
{
    GPIO_DRIVING_LEVEL0    = 0,      /**< Defines GPIO driving current as level0.  */
    GPIO_DRIVING_LEVEL1    = 1,      /**< Defines GPIO driving current as level1.  */
    GPIO_DRIVING_LEVEL2    = 2,      /**< Defines GPIO driving current as level2. */
    GPIO_DRIVING_LEVEL3    = 3       /**< Defines GPIO driving current as level3. */
} gpio_driving_level_t;
```

3.1.4 引脚上下拉 gpio_pull_status_t

该枚举定义了引脚的上下拉的值，具体定义如下：

```
typedef enum
{
    GPIO_PULL_DOWN_DISABLED    = 0,      /**< Defines GPIO pull up and pull down disable.  */
    GPIO_PULL_UP          = 1,      /**< Defines GPIO is pull up state.  */
    GPIO_PULL_DOWN            = 2,      /**< Defines GPIO is pull down state. */
} gpio_pull_status_t;
```

3.1.5 引脚数据 gpio_data_t

该枚举定义引脚的输入输出数据，具体定义如下：

```
typedef enum
{
    GPIO_DATA_LOW  = 0,             /**<  GPIO data low. */
    GPIO_DATA_HIGH = 1              /**<  GPIO data high. */
} gpio_data_t;
```

3.1.6 引脚电压能力 gpio_power_mode_t

该枚举定义了引脚的电压模式，可以配置成 1.8V 和 3.3V，具体定义如下

```
typedef enum
{
    POWER_MODE_330 = 0,
    POWER_MODE_180 = 1
} gpio_power_mode_t;
```

3.1.7 中断模式 gpio_interrupt_mode_t

该枚举定义了引脚的中断模式，具体定义如下：

```
typedef enum
{
    IRQ_TYPE_NONE       = 0x00000000,
    IRQ_TYPE_EDGE_RISING    = 0x00000001,
    IRQ_TYPE_EDGE_FALLING   = 0x00000002,
    IRQ_TYPE_EDGE_BOTH  = (IRQ_TYPE_EDGE_FALLING | IRQ_TYPE_EDGE_RISING),
    IRQ_TYPE_LEVEL_HIGH = 0x00000004,
    IRQ_TYPE_LEVEL_LOW  = 0x00000008,
} gpio_interrupt_mode_t;
```

3.2 驱动层接口

驱动层提供的接口可以通过查看 sunxi_hal_gpio.h 获取，下面介绍一些比较重要的接口。

3.2.1 配置引脚数据

配置引脚数据主要是以下两个函数，具体说明如下：

```
int hal_gpio_get_data(gpio_pin_t pin, uint32_t *data);
//获取引脚当前数据状态；pin：引脚，data：存放数据的值；返回值：成功返回0，失败返回-1
int hal_gpio_set_data(gpio_pin_t pin, uint32_t data);
//设置引脚的数据状态；pin：引脚，data：设置数据的值；返回值：成功返回0，失败返回-1
```

3.2.2 配置引脚上下拉

配置引脚的上下拉能力主要是以下两个函数，具体说明如下：

```
int hal_gpio_set_pull(gpio_pin_t pin, gpio_pull_status_t pull);
//配置引脚的上下拉能力；pin：引脚，pull：上下拉配置；返回值：成功返回0，失败返回-1
int hal_gpio_get_pull(gpio_pin_t pin, gpio_pull_status_t *pull);
//获取引脚的上下拉能力；pin：引脚，pull：上下拉配置；返回值：成功返回0，失败返回-1
```

3.2.3 配置引脚驱动能力

配置引脚的驱动能力主要是以下两个函数，具体说明如下：

```
int hal_gpio_set_driving_level(gpio_pin_t pin, gpio_driving_level_t level);
//配置引脚的驱动能力；pin：引脚，level：驱动能力等级；返回值：成功返回0，失败返回-1
int hal_gpio_get_driving_level(gpio_pin_t pin, gpio_driving_level_t *level);
//获取引脚的驱动能力；pin：引脚，level：驱动能力；返回值：成功返回0，失败返回-1
```

3.2.4 配置引脚复用功能

配置引脚的复用功能主要是以下函数，具体说明如下：

```
int hal_gpio_pinmux_set_function(gpio_pin_t pin, gpio_muxsel_t function_index);
//配置引脚的复用能力；pin：引脚，function_index：复用功能；返回值：成功返回0，失败返回-1
```

3.2.5 配置引脚电压模式

配置引脚的电压模式主要是以下函数，具体说明如下：

```
int hal_gpio_sel_vol_mode(gpio_pin_t pins, gpio_power_mode_t pm_sel);
//配置引脚的电压模式能力；pin：引脚，pm_sel：电压模式；返回值：成功返回0，失败返回-1
```

3.2.6 申请引脚中断

由于一组引脚共用一个中断号，所以引用了二级中断，当中断来临时，先获取引脚中断状态，判断是那个引脚的中断，再调用引脚的中断，所以在申请中断的时候，引脚的中断实际为二级中断，需要先获取二级中断号，然后才可以申请中断，具体步骤如下：

```
int hal_gpio_to_irq(gpio_pin_t pin, uint32_t *irq);
//获取引脚的中断号，存放在irq中；pin：引脚，irq：中断号；成功返回0，失败返回-1
int hal_gpio_irq_request(uint32_t irq, irq_handler_t hdle, unsigned long flags, void *data);
//申请中断；irq：上一步获取的中断号，hdle：中断处理函数，flags：中断类型，data：传入中断的数据
int hal_gpio_irq_enable(uint32_t irq);
//使能中断
```

3.2.7 释放引脚中断

释放引脚中断的函数刚好与申请中断相反，具体如下：

```
int hal_gpio_to_irq(gpio_pin_t pin, uint32_t *irq);
//获取引脚的中断号，存放在irq中；pin：引脚，irq：中断号；成功返回0，失败返回-1
int hal_gpio_irq_disable(uint32_t irq);
```

```
//失能中断，irq为上一步获取的中断号；成功返回0，失败返回-1
int hal_gpio_irq_free(uint32_t irq);
//释放中断，irq为上一步获取的中断号；成功返回0，失败返回-1
```

3.3 应用层接口

应用层接口的使用与驱动层用法类似，只是在使用应用层接口的时候加了一层参数过滤，具体如下所示：

3.3.1 配置引脚数据

配置引脚数据主要是以下两个函数，具体说明如下：

```
gpio_status_t drv_gpio_get_input(gpio_pin_t gpio_pin, gpio_data_t *gpio_data);
//获取引脚当前数据状态；gpio_pin：引脚，gpio_data：存放数据的值；返回值：成功返回0，失败返回负数
gpio_status_t drv_gpio_set_output(gpio_pin_t gpio_pin, gpio_data_t gpio_data);
//设置引脚的数据状态；gpio_pin：引脚，gpio_data：设置数据的值；返回值：成功返回0，失败返回负数
```

3.3.2 配置引脚上下拉

配置引脚的上下拉能力主要是以下两个函数，具体说明如下：

```
gpio_status_t drv_gpio_set_pull_status(gpio_pin_t gpio_pin, gpio_pull_status_t gpio_pull_status);
//配置引脚的上下拉能力；gpio_pin：引脚，gpio_pull_status：上下拉配置；返回值：成功返回0，失败返回负数
gpio_status_t drv_gpio_get_pull_status(gpio_pin_t gpio_pin, gpio_pull_status_t *gpio_pull_status);
//获取引脚的上下拉能力；gpio_pin：引脚，gpio_pull_status：上下拉配置；返回值：成功返回0，失败返回负数
```

3.3.3 配置引脚驱动能力

配置引脚的驱动能力主要是以下两个函数，具体说明如下：

```
gpio_status_t drv_gpio_set_driving_level(gpio_pin_t gpio_pin, gpio_driving_level_t gpio_driving_level);
//配置引脚的驱动能力；gpio_pin：引脚，gpio_driving_level：驱动能力等级；返回值：成功返回0，失败返回负数
gpio_status_t drv_gpio_get_driving_level(gpio_pin_t gpio_pin, gpio_driving_level_t *gpio_driving_level);
//获取引脚的驱动能力；gpio_pin：引脚，gpio_driving_level：驱动能力；返回值：成功返回0，失败返回负数
```

3.3.4 配置引脚复用功能

配置引脚的复用功能主要是以下函数，具体说明如下：

```
gpio_status_t drv_gpio_pinmux_set_function(gpio_pin_t gpio_pin, uint32_t function_index);
//配置引脚的复用能力；gpio_pin：引脚，function_index：复用功能；返回值：成功返回0，失败返回负数
```

3.3.5 配置引脚电压模式

配置引脚的电压模式主要是以下函数，具体说明如下：

```
gpio_status_t drv_gpio_sel_vol_mode(gpio_pin_t gpio_pin, uint32_t sel_num);
//配置引脚的电压模式能力；gpio_pin：引脚，sel_num：电压模式；返回值：成功返回0，失败返回负数
```

3.3.6 申请引脚中断

由于一组引脚共用一个中断号，所以引用了二级中断，当中断来临时，先获取引脚中断状态，判断是那个引脚的中断，再调用引脚的中断，所以在申请中断的时候，引脚的中断实际为二级中断，需要先获取二级中断号，然后才可以申请中断，具体步骤如下：

```
gpio_status_t drv_gpio_to_irq(gpio_pin_t gpio_pin, int32_t *irq);
//获取引脚的中断号，存放在irq中；gpio_pin：引脚，irq：中断号；成功返回0，失败返回负数
gpio_status_t drv_gpio_irq_request(uint32_t irq, irq_handler_t hdle, unsigned long flags, void *data);
//申请中断；irq：上一步获取的中断号，hdle：中断处理函数，flags：中断类型，data：传入中断的数据
gpio_status_t drv_gpio_irq_enable(uint32_t irq);
```

```
//使能中断
```

3.3.7 释放引脚中断

释放引脚中断的函数刚好与申请中断相反，具体如下：

```
gpio_status_t drv_gpio_to_irq(gpio_pin_t gpio_pin, int32_t *irq);
//获取引脚的中断号，存放在irq中；gpio_pin：引脚，irq：中断号；成功返回0，失败返回负数
gpio_status_t drv_gpio_irq_disable(uint32_t irq);
//失能中断，irq为上一步获取的中断号；成功返回0，失败返回负数
gpio_status_t drv_gpio_irq_free(uint32_t irq);
//释放中断，irq为上一步获取的中断号；成功返回0，失败返回负数
```

4. 模块使用 DEMO

GPIO 模块的使用 DEMO 在 source/ekernel/drivers/test/test_gpio.c，如果想要了解使用可以查看该模块。

该测试用例对应用层跟驱动层的 CPUX/CPUS 域的引脚都做了验证。

当系统进入控制台后，可以输入 drvgpio 来测试应用层的 gpio，输入 gpio 来测试驱动层的 gpio。

5. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This document neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.