君正®

T41 GPIO 寄存器操作指南

Date: 2022-05

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

Trademarks and Permissions

Disclaimer

All the deliverables and data in this folder serve only as a reference for customer development. Please read through this disclaimer carefully before you use the deliverables and data in this folder. You may use the deliverables in this folder or not. However, by using the deliverables and data in this folder, you agree to accept all the content in this disclaimer unconditional and irrevocable. If you do not find the content in this disclaimer reasonable, you shall not use the deliverables and data in this folder.

The deliverables and data in this folder are provided "AS IS" without representations, guarantees or warranties of any kind (either express or implied). To the maximum extent permitted by law, Ingenic Semiconductor Co., Ltd (Ingenic) provides the deliverables and data in this folder without implied representations, guarantees or warranties, including but not limited to implied representations, guarantees and warranties of merchantability, non-infringement, or fitness for a particular purpose. Deviation of the data provided in this folder may exist under different test environments.

Ingenic takes no liability or legal responsibility for any design and development error, incident, negligence, infringement, and loss (including but not limited to any direct, indirect, consequential, or incidental loss) caused by the use of data in this folder. Users shall be responsible for all risks and consequences caused by the use of data in this folder.

北京君正集成电路股份有限公司
地址：北京市海淀区西北旺东路 10 号院东区 14 号楼君正大厦
电话:**(86-10)56345000**
传真:**(86-10)56345001**
Http: //www.ingenic.cn

前言

概述

本文为 T41 GPIO 寄存器操作指南，帮助使用者快速了解 T 系列开发板 GPIO 寄存器在各个场景下的操作使用。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
T 系列	

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

日期	版本	修订章节
2022-05	1.0	第一次正式版本发布

目录

1 GPIO 和常用寄存器介绍 2
1.1 GPIO Group 基地址 2
1.2 常用寄存器 2
1.3 GPIO 功能设置 3
1.4 驱动能力设置 3
2 uboot 下操作 GPIO 5
2.1 GPIO 的相关函数 5
2.2 C 库函数 5
2.3 uboot 命令行 6
2.4 volatile 函数 6
3 内核中操作 GPIO 及读取寄存器 7
3.1 GPIO 的相关函数 7
3.2 volatile 函数 8
3.3 库函数 8
4 文件系统下读取 GPIO 9
4.1 devmem 读写 9
4.2 使用 GPIO 节点 10
5 应用层操作 GPIO 或者操作寄存器 12
5.1 使用 C 库函数 12
5.2 使用地址映射 12
5.3 通过节点句柄操作 13

1 GPIO 和常用寄存器介绍

1.1 GPIO Group 基地址

君正的 gpio 模块当前支持 3 组或者更多组的 gpio Group，每组 gpio 的基地址如下：

PA: 0x10010000
PB: 0x10011000
PC: 0x10012000
PD: 0x10013000

注意：每组 gpio 均对应 32 个引脚。

1.2 常用寄存器

表 1-1 常用寄存器

寄存器名称	寄存器地址	寄存器设置	寄存器清除	功能
PxINT	0x10	0x14	0x18	-
PxMSK	0x20	0x24	0x28	-
PxPAT1	0x30	0x34	0x38	-
PxPAT0	0x40	0x44	0x48	-
PxPUEN	0x110	0x114	0x118	上拉使能
PxPDEN	0x120	0x124	0x128	下拉使能
PxDSL	0x130	0x134	0x138	低 16bit 驱动能力
PxDSH	0x140	0x144	0x148	高 16bit 驱动能力

注意：

1. Px 中的 x 代表 A，B，C，D。
2. 每个功能寄存器有 3 个寄存器。
3. 寄存器设置：对固定的 bit 写 1，使能对应 pin 功能。
4. 寄存器清除：对固定的 bit 写 1，清除对应 pin 功能。

1.3 GPIO 功能设置

表 1-2　GPIO 功能描述

INT	MASK	PAT1	PAT0	端口描述
1	0	0	0	低电平触发中断
1	0	0	1	高电平触发中断
1	0	1	0	下降沿触发中断
1	0	1	1	上升沿触发中断
1	1	0	0	低电平触发中断，如果中断屏蔽，记录标志位
1	1	0	1	高电平触发中断，如果中断屏蔽，记录标志位
1	1	1	0	下降沿触发中断，如果中断屏蔽，记录标志位
1	1	1	1	上升沿触发中断，如果中断屏蔽，记录标志位
0	0	0	0	功能 0
0	0	0	1	功能 1
0	0	1	0	功能 2
0	0	1	1	功能 3
0	1	0	0	GPIO 输出 0
0	1	0	1	GPIO 输出 1
0	1	1	？	GPIO 输入

如表 1-2，使用 0x10，0x20，0x30，0x40 寄存器，可以设置某个引脚的功能，或者读取对应引脚的功能信息和电平信息。

1.4 驱动能力设置

T21/T31 驱动能力相关寄存器:

图 1-3 驱动能力设置 1

PxDSL																												0x130	
Bit	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
	DSL15[1:0]		DSL14[1:0]		DSL13[1:0]		DSL12[1:0]		DSL11[1:0]		DSL10[1:0]		DSL09[1:0]		DSL08[1:0]		DSL07[1:0]		DSL06[1:0]		DSL05[1:0]		DSL04[1:0]		DSL03[1:0]		DSL02[1:0]		DSL01[1:0]		DSL00[1:0]	
RST	0	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Bits	Name	Description	R/W
31:0	DSL n [1:0]	Where n = 0 ~ 15 and DSL n = DSL0 ~ DSL15. DSL n[1:0] is used to select drive strength of the corresponding PxDSL[2n+1 :2n]. 00: 2mA 01: 4mA 10: 8mA 11: 12mA	RW

图 1-4 驱动能力设置 2

PxDSH																												0x140	
Bit	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
	DSH15[1:0]		DSH14[1:0]		DSH13[1:0]		DSH12[1:0]		DSH11[1:0]		DSH10[1:0]		DSH09[1:0]		DSH08[1:0]		DSH07[1:0]		DSH06[1:0]		DSH05[1:0]		DSH04[1:0]		DSH03[1:0]		DSH02[1:0]		DSH01[1:0]		DSH00[1:0]	
RST	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Bits	Name	Description	R/W
31:0	DSH n [1:0]	Where n = 0 ~ 15 and DSH n = DSH0 ~ DSH15. DSH n[1:0] is used to select drive strength of the corresponding PxDSH[2n+1 :2n]. 00: 2mA 01: 4mA 10: 8mA 11: 12mA	RW

如图 1-3 和图 1-4，驱动能力是由 PxDSL 和 PxDSH 两组寄存器设置，其中 PxDSL 对应一组 gpio 的低 16bit， PxDSH 对应一组 gpio 的高 16bit。

2 uboot 下操作 GPIO

2.1 GPIO 的相关函数

对应函数在 uboot 的位置：opensource/uboot/drivers/gpio/jz_gpio_common.c

```
int gpio_request(unsigned gpio, const char *label)      //gpio 申请
int gpio_free(unsigned gpio)                            //gpio 释放
int gpio_set_value(unsigned gpio, int value)            //设置 gpio 的值
int gpio_get_value(unsigned gpio)                       //获取 gpio 的值
int gpio_direction_input(unsigned gpio)                 //设置 gpio 为输入
int gpio_direction_output(unsigned gpio, int value)    //设置 gpio 为输出
void gpio_enable_pull_up(unsigned gpio)           //打开 gpio 的上拉使能
void gpio_disable_pull_up(unsigned gpio)          //关闭 gpio 的上拉使能
void gpio_enable_pull_down(unsigned gpio)         //打开 gpio 的下拉使能
void gpio_disable_pull_down(unsigned gpio)        //关闭 gpio 的下拉使能
void gpio_set_func(enum gpio_port n, enum gpio_function func, unsigned
int pins)                                         //设置 gpio 功能
void gpio_set_driver_strength(enum gpio_port n, unsigned int pins, int
ds)                                              //设置 gpio 驱动能力
int gpio_get_flag(unsigned int gpio)             //设置 gpio 标志
int gpio_clear_flag(unsigned gpio)               //获取 gpio 标志
```

2.2 C 库函数

头文件：#include<asm/io.h>

```
readb(addr)
readw(addr)
readl(addr)

writeb(b, addr)
writew(b, addr)
writel(b, addr)
```

2.3 uboot 命令行

```
md 0x10010140              //读取 0x10010140 寄存器的值
mw 0x10010140  0x1000   //往寄存器 0x10010140 写入值 0x1000
```

2.4 volatile 函数

```
*(volatile unsigned int *)0x10010140 = 0x1000   //写入
Value = *(volatile unsigned int *)0x10010140    //读取
```

3 内核中操作 GPIO 及读取寄存器

3.1 GPIO 的相关函数

头文件:#include <linux/gpio.h>

T21/T31 及 T31 之后芯片相关函数：

```
int gpio_request(unsigned gpio, const char *label);   //gpio 申请
 void gpio_free(unsigned gpio);                        //gpio 释放
int gpio_direction_input(unsigned gpio);              //设置 gpio 为输入
 int gpio_direction_output(unsigned gpio, int value); //设置 gpio 为输出
int gpio_get_value(unsigned int gpio);                //获取 gpio 的值
void gpio_set_value(unsigned int gpio, int value);    //获取 gpio 的值
int gpio_to_irq(unsigned int gpio);                   //设置 gpio 为中断
```

T40 及 T40 之后芯片相关函数：

```
int gpio_request(unsigned gpio, const char *label)    //gpio 申请
void gpio_free(unsigned gpio)                         //gpio 释放
 int gpio_export(unsigned gpio, bool direction_may_change)  //export
gpio
void gpio_unexport(unsigned gpio)                     //unexport gpio
int __gpio_get_value(unsigned gpio)//获取 gpio 的值
void __gpio_set_value(unsigned gpio, int value)       //设置 gpio 的值
int gpio_sysfs_set_active_low(unsigned gpio, int value)
static inline int gpio_direction_output(unsigned gpio, int value)//设置
```

```
gpio 为输出
static inline int gpio_direction_input(unsigned gpio) //设置 gpio 为输入
 int __gpio_to_irq(unsigned gpio)                     //设置 gpio 为中断
```

3.2 volatile 函数

```
*(volatile unsigned int *)0x10010140 = 0x1000   //写入
Value = *(volatile unsigned int *)0x10010140    //读取
```

3.3 库函数

```
static inline void outb(u8 value, unsigned long addr)
static inline void outw(u16 value, unsigned long addr)
static inline void outl(u32 value, unsigned long addr)

static inline u8 inb(unsigned long addr)
static inline u8 inw(unsigned long addr)
static inline u8 inl(unsigned long addr)
```

4 文件系统下读取 GPIO

4.1 devmem 读写

- devmem 写

  设置 PB15 为输出(out 0)：

```
devmem 0x10011018 32 0x8000
devmem 0x10011024 32 0x8000
devmem 0x10011038 32 0x8000
devmem 0x10011048 32 0x8000
```

  设置 PB15 为输出(out 1)：

```
devmem 0x10011018 32 0x8000
devmem 0x10011024 32 0x8000
devmem 0x10011038 32 0x8000
devmem 0x10011044 32 0x8000
```

- devmem 读

  读取状态寄存器：

```
devmem 0x10011010
devmem 0x10011020
devmem 0x10011030
devmem 0x10011040
```

4.2 使用 GPIO 节点

4.2.1 申请 GPIO 的节点

在操作 sysfs GPIO 之前需要对其进行申请。值得注意的是，由于申请 sysfs GPIO 会在内核 request_gpio，因此在内核中已经申请过的 GPIO 在 sysfs GPIO 再次申请会失败。

申请/释放 GPIO 方法如下：

```
$ cd /sys/class/gpio
$ echo [gpio_num] > export         #申请 GPIO
$ echo [gpio_num] > unexport       #释放 GPIO
```

注：gpio_num 即 GPIO 号。计算公式为：

```
PA(n) = 0 * 32 + n
PB(n) = 1 * 32 + n
PC(n) = 2 * 32 + n
...
```

例如：申请 PB(10) = 1 * 32 + 10 = 42

```
$ echo 42 > export
```

申请后在/sys/class/gpio 目录下即会出现 gpio42 目录。

```
$ echo 42 > unexport
```

释放后 gpio42 目录也会消失。释放后的 GPIO 状态并不会恢复，会保持申请时的状态（电平等）。

4.2.2 设置输入/输出方式

在申请 GPIO 后，进入 gpioN 目录，例如 gpio42，进行如下操作：

```
$ echo out > direction             #设置 PB10 为输出模式
$ echo in > direction              #设置 PB10 为输入模式
```

4.2.3 设置有效电平

gpioN 目录下有 active_low 节点，表示当前 GPIO 的有限电平，默认为 0，其

意义为，当输入/输出 value 为 0 时，GPIO 为低电平，当输入/输出 value 为 1 时，GPIO 为高电平。同样的，active_low 为 1 时，当输入/输出 value 为 0 时，GPIO 为高电平；当输入/输出 value 为 1 时，GPIO 为低电平。 也就是说，GPIO 的真实电平=value^active_low。

```
$ echo 0 > active_low        #value 是 0,表示低电平；value 是 1,表示高电平
$ echo 1 > active_low        #value 是 1,表示低电平；value 是 0,表示高电平
```

4.2.4 输入/输出

gpioN 目录下有 value 节点，表示 gpioN 的电平：当 GPIO 为输入模式时，读取到 value 的值异或 active_low 即为 GPIO 的电平；当 GPIO 为输出模式时，写入到 value 的值异或 active_low 即为 GPIO 的输出电平。

```
$ cat value                 #读取电平（输入模式）
$ echo 0 > value            #设置电平（输出模式）
```

5 应用层操作 GPIO 或者操作寄存器

5.1 使用 C 库函数

头文件: #include <stdlib.h>

```
int system(const char * string);
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg,..., char * const envp[]);
```

5.2 使用地址映射

```
  #include<sys/mman.h>
 void *mmap(void *start, size_t length, int prot, int flags, int fd, off_t
offset);
 int munmap(void *start, size_t length);
```

使用示例:

```
1.  static int regrw(uint32_t addr, uint32_t *value, int is_write)
2.  {
3.          void *map_base, *virt_addr;
4.          off_t target;
5.          unsigned page_size, mapped_size, offset_in_page;
6.          int fd, width = 32, ret = 0;
7.
8.          target = addr;
9.          fd = open("/dev/mem", is_write ? (O_RDWR | O_SYNC) : (O_RDONLY | O_SYN
    C));
10.         if (fd < 0) {
11.             IMP_LOG_ERR(TAG, "open /dev/mem failed\n");
```

```
            return -1;
        }

        mapped_size = page_size = getpagesize();
        offset_in_page = (unsigned)target & (page_size - 1);
        if (offset_in_page + width > page_size) {
            /* This access spans pages.
             * Must map two pages to make it possible: */
            mapped_size *= 2;
        }
        map_base = mmap(NULL,
                        mapped_size,
                        is_write ? (PROT_READ | PROT_WRITE) : PROT_READ,
                        MAP_SHARED,
                        fd,
                        target & ~(off_t)(page_size - 1));
        if (map_base == MAP_FAILED) {
            IMP_LOG_ERR(TAG, "mmap failed\n");
          ret = -1;
            goto close_fd;
        }

        virt_addr = (char*)map_base + offset_in_page;

        if (!is_write)
           *value = *(volatile uint32_t*)virt_addr;
       else
            *(volatile uint32_t*)virt_addr = *value;

       if (munmap(map_base, mapped_size) == -1)
            IMP_LOG_ERR(TAG, "munmap failed\n");

close_fd:
       close(fd);

       return ret;
}
```

5.3 通过节点句柄操作

此操作方法类似于调用 system 函数往节点去写内容。

示例：

```
fd = open("/sys/class/gpio/export", O_WRONLY);
    if(fd < 0) {
        IMP_LOG_DBG(TAG, "open /sys/class/gpio/export error !");
        return -1;
    }

    write(fd, "79", 2);
    write(fd, "80", 2);

    close(fd);

    fd79 = open("/sys/class/gpio/gpio79/direction", O_RDWR);
    if(fd79 < 0) {
        IMP_LOG_DBG(TAG, "open /sys/class/gpio/gpio79/direction error !");
        return -1;
    }

    fd80 = open("/sys/class/gpio/gpio80/direction", O_RDWR);
    if(fd80 < 0) {
        IMP_LOG_DBG(TAG, "open /sys/class/gpio/gpio80/direction error !");
        return -1;
    }

    write(fd79, "out", 3);
    write(fd80, "out", 3);

    close(fd79);
    close(fd80);
```