CMT4522

GPIO Application Note

Version 1.0

Author: HOERF

Security: Public

Date: 2021.01

Revision History

Revision	Author	Date	Description
V1.0		2021.01.25	This document is used for 4520/4550/4522/4552.

目录

图表目录

1 简介

GPIO，全称 General-Purpose Input/Output（通用输入输出），是一种软件运行期间能够动态配置和控制的通用引脚。

不同型号的芯片支持的 GPIO 数量有差异。

以 QFN32 为例，GPIO 上电默认属性如下表：

QFN32	Default mode	Default IN_OUT	IRQ/ Wakeup	FULLMUX	ANA	KSCAN
P00	GPIO	IN	√	√		mk_in[0]
P01	GPIO	IN	√			mk_out[0]
P02	SWD_IO	OUT	√	√		mk_in[1]
P03	SWD_CLK	IN	√	√		mk_out[1]
P07	GPIO	IN	√	√		mk_in[10]
TEST_MODE						
P09	GPIO	IN	√	√		mk_out[4]
P10	GPIO	IN	√	√		mk_in[4]
P11	GPIO	IN	√	√	√	mk_out[11]
P14	GPIO	IN	√	√	√	mk_out[2]
P15	GPIO	IN	√	√	√	mk_in[2]
P16	XTALI(ANA)	ANA			√	mk_out[10]
P17	XTALO(ANA)	ANA			√	mk_out[9]
P18	GPIO	IN	√	√	√	mk_in[5]
P20	GPIO	IN	√	√	√	mk_out[5]
P23	GPIO	IN	√	√	√	mk_in[6]
P24	GPIO	IN	√	√	√	mk_out[3]
P25	GPIO	IN	√	√	√	mk_in[3]
P26	GPIO	IN	√	√		mk_out[8]
P27	GPIO	IN	√	√		mk_in[9]
P31	GPIO	IN	√	√		mk_out[7]
P32	GPIO	IN	√	√		mk_in[7]
P33	GPIO	IN	√	√		mk_out[6]
P34	GPIO	IN	√	√		mk_in[8]

表 1：GPIO 上电默认属性配置

1.1 特殊 IO

1.1.1 TEST_MODE

TEST_MODE 有特殊用途，和 P24、P25 一起配置芯片所处状态。
因此，应用代码中不能使用 TEST_MODE，另外硬件电路上 TEST_MODE、P24、P25 需要支持的组合有{0,*,*}和{1,0,0}，前者芯片处于 Normal Mode 可运行程序，后者芯片处于 Program Mode 可烧录程序。

TEST_MODE	P24	P25	MODE
0	*	*	Normal Mode Program Mode
1	0	0	Program Mode
1	0	1	Scan Mode
1	1	0	Bist Mode

表 2：GPIO MODE 选择

Program Mode 有两种方式可以进入：

- TEST_MODE=0，CMOSTEK 需先运行在单线烧录阶段(UDLL48)或双线烧录阶段(UXTDWU)，握手成功后芯片切换到 Program Mode。
- TEST_MODE=1、P24=0、P25=0。芯片复位后检测到上述电平后进入 Program Mode。

1.1.2 P16、P17

P16、P17 默认做模拟口，接晶振、电容组成振荡电路，其中 P16 为 XTALI，P17 为 XTALO。
目前，应用代码不能使用 P16、P17 做其他用途。即使系统中采用 RC 32K，P16、P17 也不可作其他用途。
以后 P16、P17 会支持其他功能，比如 GPIO、IOMUX 等，到时文档会更新。

1.1.3 P1

P1 不支持 FULLMUX 功能，FULLMUX 是指将 GPIO 复用为其他模块引脚。

1.1.4 P2、P3

P2(SDW_IO)、P3(SDW_CLK)可以接调试器，接调试器调试程序时，这两个 IO 口的非调试功能会受影响。
因此使用 P2、P3 时，硬件不要接调试器。

1.2 GPIO 模式

GPIO 模式是最常用的模式，可配置为输出并输
出高低电平，可以配置为输入读取外部的高低电平。
当配置为输入时，支持中断和唤醒。

1.2.1 GPIO 输出

配置相应 GPIO 方向寄存器为输出，向输出寄存器写 1 或 0，即可输出高或低电平。

1.2.2 GPIO 输入

配置相应 GPIO 方向寄存器为输入，读取输入寄存器的值，即可获取当前 GPIO 的电平状态。
如使用中断，需要打开 GPIO 中断使能功能，并配置中断产生条件。
如使用唤醒，需要打开 GPIO 唤醒使能功能，并配置唤醒产生条件。

1.2.3 GPIO retention

当 GPIO 做输出时，可配置 retention 功能。retention 默认是关闭的。
retention 打开时，系统休眠时，GPIO 的输出特性和输出值保持不表。retention 关闭时，系统休眠时，GPIO 会恢复默认输入态。

如果没有 retention，系统休眠后，引脚外部电路的高低状态保持是根据引脚的输入态加上下拉电阻来实现的，此时驱动能力较弱。retention 就是为了增加驱动能力新增的功能。

比如，P00 运行时配置为 GPIO 输出且输出 1，如系统进入休眠时候，该 GPIO 会变为输入态，此时不会输出 1。如果想让该 GPIO 在休眠时也保持输出 1 这种状态，那么需要在休眠前配置该 GPIO 的 retention 功能。

1.2.4 GPIO 上下拉电阻

每个 GPIO 支持四种上下拉电阻配置：

- 浮空:高阻态。
- 强上拉:上拉到 AVDD33，高电平，驱动电流大。上拉电阻 150kΩ 欧姆。
- 弱上拉:上拉到 AVDD33，高电平，驱动电流小。上拉电阻 1MΩ 欧姆。
- 下拉:下拉到地，低电平，下拉电阻 150kΩ 欧姆。

上下拉电阻硬件默认值：

- P03、P24、P25：下拉。
- 其他 GPIO：浮空。

1.2.5 中断和唤醒

除 TEST_MODE、P16、P17 之外的所有 GPIO 支持中断和唤醒。
中断支持电平触发和边沿触发，唤醒支持边沿触发。

注意事项：GPIO 做唤醒源使用时，必须配置该
引脚的内部上拉电阻或下拉电阻，不能是高阻态。

1.3 FULLMUX 模式

除 TEST_MODE、P16、P17、P1 之外的其他 GPIO 都支持 GPIO FULLMUX 功能，可根据应用 GPIO 配置为 UART，I2C、PWM 等功能。

比如，在烧录模式下，UART 用到的就是 P9(TX)，P10(RX)。这里的 P9、P10 就是 GPIO FULLMUX 功能。在应用程序中，我们可以将其他 GPIO 复用为 UART 功能，也可以将 P9、P10 复用为 PWM 等其他功能。GPIO 和复用关系可以灵活配置的。

FULLMUX 配置含义说明：

FULLMUX define	FULLMUX value	FULLMUX description
FMUX_IIC0_SCL	0	I2C0 时钟引脚
FMUX_IIC0_SDA	1	I2C0 数据引脚
FMUX_IIC1_SCL	2	I2C1 时钟引脚
FMUX_IIC1_SDA	3	I2C1 数据引脚
FMUX_UART0_TX	4	UART0 发送引脚
FMUX_UART0_RX	5	UART0 接收引脚
FMUX_RF_RX_EN	6	RF 接收功能调试引脚
FMUX_RF_TX_EN	7	RF 发送功能调试引脚
FMUX_UART1_TX	8	UART1 发送引脚
FMUX_UART1_RX	9	UART1 接收引脚
FMUX_PWM0	10	PWM 通道 0
FMUX_PWM1	11	PWM 通道 1
FMUX_PWM2	12	PWM 通道 2
FMUX_PWM3	13	PWM 通道 3
FMUX_PWM4	14	PWM 通道 4
FMUX_PWM5	15	PWM 通道 5
FMUX_SPI_0_SCK	16	SPI0 时钟引脚
FMUX_SPI_0_SSN	17	SPI0 片选引脚
FMUX_SPI_0_TX	18	SPI0 发送引脚
FMUX_SPI_0_RX	19	SPI0 接收引脚

FMUX_SPI_0_SCK	20	SPI1 时钟引脚
FMUX_SPI_1_SSN	21	SPI1 片选引脚
FMUX_SPI_1_TX	22	SPI1 发送引脚
FMUX_SPI_1_RX	23	SPI1 接收引脚
FMUX_CHAX	24	旋转编码器 CHAX 引脚
FMUX_CHBX	25	旋转编码器 CHBX 引脚
FMUX_CHIX	26	旋转编码器 CHIX 引脚
FMUX_CHAY	27	旋转编码器 CHAY 引脚
FMUX_CHBY	28	旋转编码器 CHBY 引脚
FMUX_CHIY	29	旋转编码器 CHIY 引脚
FMUX_CHAZ	30	旋转编码器 CHAZ 引脚
FMUX_CHBZ	31	旋转编码器 CHBZ 引脚
FMUX_CHIZ	32	旋转编码器 CHIZ 引脚
FMUX_CLK1P28M	33	DMIC 时钟引脚
FMUX_ADCC	34	DMIC 数据引脚
FMUX_ANT_SEL_0	35	天线选择 0，定位用
FMUX_ANT_SEL_1	36	天线选择 1，定位用
FMUX_ANT_SEL_2	37	天线选择 2，定位用

表 3：GPIO FULLMUX 选择

1.4 ANALOG 模式

只有 P11、P14、P15、P16、P17、P18、P20、P23、P24、P25 支持模拟功能。

- 32.768K 晶振振荡电路：P16、P17 接电容、32.768K 晶振组成振荡电路，P16、P17 不能做其他用途。
- VOICE：VOICE 支持 DMIC 和 AMIC。当使用 AMIC 电路时，用到的引脚有 P18(pga+)、P20(pga-)、P15(micphone bias)、P23(micphone bias reference voltage)，其中 P23 是可选的。
- ADC:采集引脚上的电压，单端支持的引脚有 P11、P23、P24、P14、P15、P20，差分支持的引脚有 P18P25、P23P11、P14P24、P20P15。

1.5 KSCAN 模式

当键盘中按键数量较多时，为了减少 I/O 口的占用，通常将按键排列成矩阵形式。在矩阵式键盘中，每条水平线和垂直线在交叉处不直接连通，而是通过一个按键加以连接。

比如：4*4 矩阵按键支持 16 个按键，比常规按键多出一倍的按键数量。在需要的键数比较多时，采用矩阵法来做键盘更合理。
CMT45**内置硬件矩阵电路，使用简单，处理高效。

比如：有一 4*4 矩阵按键，KSCAN 使用的引脚如下：

row	col	KSCAN	GPIO
row0		mk_in[0]	P0
row1		mk_in[1]	P2
row2		mk_in[2]	P15
row3		mk_in[3]	P25
	col0	mk_out[0]	P1
	col1	mk_ out [1]	P3
	col2	mk_ out [2]	P14
	col3	mk_ out [3]	P24

图 1：4*4 矩阵按键引脚分配图

KSCAN mk_in、mk_out 和 GPIO 的对应关系见表 1：GPIO 上电默认属性配置。

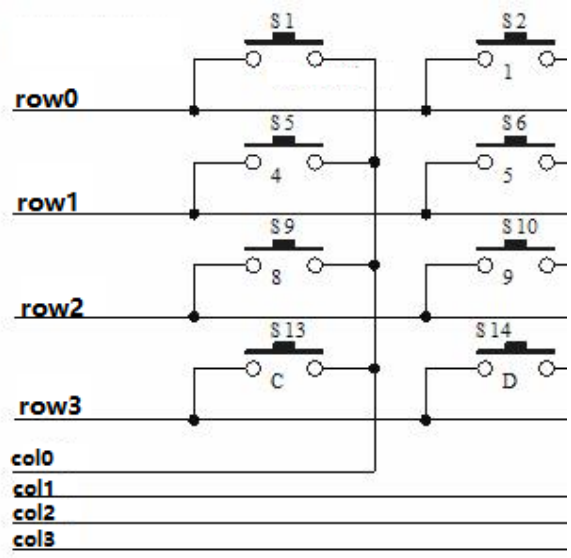

图 2：4*4 矩阵按键示意图

Key	Row	Col
S1	0	0
S2	0	1
S3	0	2
S4	0	3
S5	1	0
S6	1	1
S7	1	2
S8	1	3
S9	2	0
S10	2	1
S11	2	2
S12	2	3
S13	3	0
S14	3	1
S15	3	2
S16	4	3

表 4：4*4 矩阵按键对应图

当 KSCAN 按键时，会产生中断，将按键行列信息 mk_in[m]和 mk_out[n]中的 m、n 告知上层。

2 GPIO 典型应用

2.1 GPIO 输出

配置对应 GPIO 方向寄存器(swporta_ddr)为输出，设置输出寄存器(swporta_dr)为 0 或 1。
驱动有对应的 API，直接调用即可。
比如：将 P0 设置为输出，并不停输出 0 和 1。

```
//4520/4550
static void simple_code(void)
{
    int32_t pin = P0;
    gpio_pin_handle_t pin_p = NULL;
    int32_t ret;

    pin_p = csi_gpio_pin_initialize(pin, NULL);
    ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_OUTPUT);
    while(1)
    {
        csi_gpio_pin_write(pin_p,1);
        csi_gpio_pin_write(pin_p,0);
    }
}
```

```
//4522/4552
static void simple_code(void)
{
        gpio_pin_e pin = P0;
        hal_gpio_pin_init(pin,GPIO_OUTPUT);
        while(1)
        {
            hal_gpio_fast_write(pin,1);
            hal_gpio_fast_write(pin,0);
        }
}
```

2.2 GPIO 输入

配置相应 GPIO 方向寄存器(swporta_ddr)为输入，读取输入寄存器(swporta_ddr)，即可获取当前 GPIO 的电平状态。
比如将 P0 配置为输入，并读取当前 GPIO 电平状态。

```
//4520/4550
static void simple_code(void)
{
```

```
    int32_t pin = P0;
    gpio_pin_handle_t pin_p = NULL;
    int32_t ret;
    bool value;

    pin_p = csi_gpio_pin_initialize(pin, NULL);
    ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_INPUT);
    ret = csi_gpio_pin_read(pin_p, &value);
    printf("pin:%d value:%d\n",pin,value);
}
```

```
//4522/4552
static void simple_code(void)
{
        gpio_pin_e pin = P0;
        bool value;

        hal_gpio_pin_init(pin,GPIO_INPUT);
        value = hal_gpio_read(pin);
        LOG("pin:%d value:%d\n",pin,value);
}
```

2.3 GPIO retention

当 GPIO 配置 GPIO 输出时，当系统休眠后，GPIO 输出信息将会丢失。如果想在系统休眠后，仍保持 GPIO 输出状态并保持输出的高低电平，需要使用 GPIO retention 功能。
比如：将 P0 设置为输出，并期望当系统休眠时可以保持。

```
//4520/4550
static void simple_code(void)
{
    int32_t pin = P0;
    gpio_pin_handle_t pin_p = NULL;
    int32_t ret;

    pin_p = csi_gpio_pin_initialize(pin, NULL);
    ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_OUTPUT);
    csi_gpio_pin_write(pin_p,1);
    phy_gpioretention_register(pin); //enable this pin retention
    //phy_gpioretention_unregister(pin);// disable this pin retention
}
```

```
//4522/4552
static void simple_code(void)
{
```

```
    gpio_pin_e pin = P0;
    hal_gpioretention_register(pin);//enable this pin retention
    //hal_gpioretention_unregister(pin);//disable this pin retention
    hal_gpio_write(pin,1);
}
```

2.4 GPIO 上下拉电阻

每个 GPIO 支持四种上下拉配置：悬空、强上拉、上拉、下拉。
比如：将 P0 配置为输入，配置强上拉，并读取当前 GPIO 电平状态。

```
//4520/4550
static void simple_code(void)
{
    int32 pin = P0;
    gpio_pin_handle_t pin_p = NULL;
    gpio_pupd_e type = GPIO_PULL_UP_S;
    bool value;
    int32 ret;

    pin_p = csi_gpio_pin_initialize(pin, NULL);
    phy_gpio_pull_set(pin,type);
    ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_INPUT);
    ret = csi_gpio_pin_read(pin_p, &value);
    printf("pin:%d value:%d\n",pin,value);
}
```

```
//4522/4552
static void simple_code(void)
{
        gpio_pin_e pin = P0;
        volatile bool value;

        hal_gpio_pull_set(pin,GPIO_PULL_UP_S);
        hal_gpio_pin_init(pin,GPIO_INPUT);
        value = hal_gpio_read(pin);
        LOG("pin:%d value:%d\n",pin,value);
}
```

2.5 中断和唤醒

使用 GPIO 中断时，需要配置 GPIO 中断产生条件。中断产生后，GPIO 驱动会响应中断并调用用户配置的回调函数。
使用 GPIO 唤醒时，当系统进入休眠前，根据当前 GPIO 的电平状态设置唤醒系统的条件，当该条件产生时，系统唤醒并会调用用户配置的回调函数。

比如：将 P0 配置为输入，支持中断和唤醒，支持上升沿下降沿触发。

```
//4520/4550 int
void gpio_event_cb(int idx)
{
	printf("pin:%d\n",idx);
}

static void simple_code(void)
{
	int32 pin = P0;
	gpio_pin_handle_t pin_p = NULL;
	int32 ret;
	bool value;

	drv_pinmux_config(pin, PIN_FUNC_GPIO);
	pin_p = csi_gpio_pin_initialize(pin, gpio_event_cb);
	ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_INPUT);
	ret = csi_gpio_pin_read(pin_p, &value);

	if(value == 0)
		ret = csi_gpio_pin_set_irq(pin_p, GPIO_IRQ_MODE_RISING_EDGE, 1);
	else
		ret = csi_gpio_pin_set_irq(pin_p, GPIO_IRQ_MODE_FALLING_EDGE, 1);
......
}

//4520/4550 wakeup
static void simple_code(void)
{
		int32 pin = P0;
		gpio_pin_handle_t pin_p = NULL;
		int32 ret;
		bool value;

		drv_pinmux_config(pin, PIN_FUNC_GPIO);
		pin_p = csi_gpio_pin_initialize(pin, gpio_event_cb);
		ret = csi_gpio_pin_config_direction(pin_p, GPIO_DIRECTION_INPUT);
		ret = csi_gpio_pin_read(pin_p, &value);

		if(value == 0)
			phy_gpio_wakeup_set(pin,POL_RISING);
		else
			phy_gpio_wakeup_set(pin,POL_FALLING);
```

```
.....
}
```

```
//4522/4552
void posedge_int_wakeup_cb(GPIO_Pin_e pin,IO_Wakeup_Pol_e type)
{
        if(type == POSEDGE)
        {
            LOG("int or wakeup(pos):gpio:%d type:%d\n",pin,type);
        }
        else
        {
            LOG("error\n");
        }
}

void negedge_int_wakeup_cb(GPIO_Pin_e pin,IO_Wakeup_Pol_e type)
{
        if(type == NEGEDGE)
        {
            LOG("int or wakeup(neg):gpio:%d type:%d\n",pin,type);
        }
        else
        {
            LOG("error\n");
        }
}
static void simple_code(void)
{
        gpio_pin_e pin = P0;
        hal_gpioin_register(pin,posedge_int_wakeup_cb,negedge_int_wakeup_cb);
}
```

2.6 FULLMUX 模式

使用 GPIO FULLMUX 时，配置 FULLMUX 功能并打开 FULLMUX 使能。不使用时，一定要关闭其 FULLMUX 使能。

比如将 P9、P10 复用为 UART，其中 P9 做 TX、P10 做 RX，波特率为 115200，使用 UART0。

```
//4520/4550
#define CONSOLE_UART_IDX 0
#define CONSOLE_TXD                 P9
#define CONSOLE_RXD                 P10
#define CONSOLE_TXD_FUNC             FMUX_UART0_TX
```

```
#define CONSOLE_RXD_FUNC                    FMUX_UART0_RX
......
drv_pinmux_config(CONSOLE_TXD, CONSOLE_TXD_FUNC);
drv_pinmux_config(CONSOLE_RXD, CONSOLE_RXD_FUNC);
......
console_init(CONSOLE_UART_IDX, 115200, 0);
......
```

```
//4522/4552
void dbg_printf_init(void)
{
        uart_Cfg_t cfg =
        {
        .tx_pin = P9,
        .rx_pin = P10,
        .rts_pin = GPIO_DUMMY,
        .cts_pin = GPIO_DUMMY,
        .baudrate = 115200,
        .use_fifo = TRUE,
        .hw_fwctrl = FALSE,
        .use_tx_buf = FALSE,
        .parity      = FALSE,
        .evt_handler = NULL,
        };
        hal_uart_init(cfg, UART0);
}

......
    hal_gpio_fmux_set(pcfg->tx_pin, fmux_tx);//P9 FMUX_UART0_TX
    hal_gpio_fmux_set(pcfg->rx_pin, fmux_rx);//P10 FMUX_UART0_RX
......
```

2.7 ANALOG 模式

详见《CMT4522_ADC_Application_Note》

2.8 KSCAN 模式

使用 KSCAN 模式时，需要配置：

- 上下拉电阻
- 行用到的引脚配置为 mk_in，列用到的引脚配置为 mk_out
- 配置 KSCAN 相关寄存器

KSCAN 驱动已将上述操作封装 API，直接配置相应的引脚和回调函数即可。

比如：定义一个 2*2 的矩阵按键，其中行使用 P23、P18，列使用 P24、P11。
当有按键按下或按键释放时，回调处理函数是 kscan_evt_handler。

```
//4520/4550
#define NUM_KEY_ROWS    2
#define NUM_KEY_COLS     2

KSCAN_ROWS_e rows[NUM_KEY_ROWS] = { KEY_ROW_P23,KEY_ROW_P18};
KSCAN_COLS_e cols[NUM_KEY_COLS] = { KEY_COL_P24,KEY_COL_P11};


static void kscan_evt_handler(kscan_Evt_t* evt)
{
    printf("kscan_evt_handler\n");
    printf("num: ");
    printf("%d",evt->num);
    printf("\n");

    for(uint8_t i=0;i<evt->num;i++){
        printf("index: ");
        printf("%d",i);
        printf(",row: ");
        printf("%d",evt->keys[i].row);
        printf(",col: ");
        printf("%d",evt->keys[i].col);
        printf(",type: ");
        printf("%s",evt->keys[i].type == KEY_PRESSED ? "pressed":"released");
        printf("\n");
    }
}

void KSCAN_Init(void)
{
    printf("KSCAN_Init\n");
    kscan_Cfg_t cfg;
    cfg.ghost_key_state = NOT_IGNORE_GHOST_KEY;
    cfg.key_rows = rows;
    cfg.key_cols = cols;
```

```
    cfg.interval = 50;
    cfg.evt_handler = kscan_evt_handler;
    hal_kscan_init(cfg, 0, 0);
}
```

```
//4522/4552
#define NUM_KEY_ROWS    2
#define NUM_KEY_COLS    2

KSCAN_ROWS_e rows[NUM_KEY_ROWS] = {KEY_ROW_P23,KEY_ROW_P18};
KSCAN_COLS_e cols[NUM_KEY_COLS] = {KEY_COL_P24,KEY_COL_P11};

static void kscan_evt_handler(kscan_Evt_t* evt)
{
        LOG("\nkscan_evt_handler\n");
        LOG("num: %d\n",evt->num);

        for(uint8_t i=0; i<evt->num; i++)
        {
                LOG("index: ");
                LOG("%d",i);
                LOG(",row: ");
                LOG("%d",evt->keys[i].row);
                LOG(",col: ");
                LOG("%d",evt->keys[i].col);
                LOG(",type: ");
                LOG("%s",evt->keys[i].type == KEY_PRESSED ? "pressed":"released");
                LOG("\n");
        }
}

static void simple_doce(void)
{
        kscan_Cfg_t cfg;
        cfg.ghost_key_state = NOT_IGNORE_GHOST_KEY;
        cfg.key_rows = rows;
        cfg.key_cols = cols;
        cfg.interval = 50;
        cfg.evt_handler = kscan_evt_handler;
        hal_kscan_init(cfg, task_id, KSCAN_WAKEUP_TIMEOUT_EVT);
}
```