

---

# 通用 GPIO 设备应用笔记

---

**RT-THREAD 文档中心**

上海睿赛德电子科技有限公司版权 ©2019



[WWW.RT-THREAD.ORG](http://WWW.RT-THREAD.ORG)

Friday 28<sup>th</sup> September, 2018

# 目录

目录	i
1 本文的目的和结构	1
1.1 本文的目的和背景	1
1.2 本文的结构	1
2 问题阐述	1
3 问题的解决	2
3.1 准备和配置工程	3
3.2 GPIO 输出配置	4
3.3 GPIO 输入配置	6
3.4 GPIO 中断配置	7
3.5 I/O 设备管理框架和通用 GPIO 设备的联系	9
4 参考	9
4.1 本文所有相关的 API	9
4.1.1. API 列表	10
4.1.2. 核心 API 详解	10
4.1.2.1. rt_pin_mode()	10
4.1.2.2. rt_pin_write()	11
4.1.2.3. rt_pin_read()	12
4.1.2.4. rt_pin_attach_irq()	12
4.1.2.5. rt_pin_detach_irq()	14
4.1.2.6. rt_pin_irq_enable()	14

!!! abstract “摘要” 本应用笔记描述了如何使用 RT-Thread 的通用 GPIO 设备驱动，包括驱动的配置、相关 API 的应用。并给出了在正点原子 STM32F4 探索者开发板上验证的代码示例。

## 1 本文的目的和结构

### 1.1 本文的目的和背景

为了给用户提供操作 GPIO 的通用 API，方便应用程序开发，RT-Thread 中引入了通用 GPIO 设备驱动。并提供类似 Arduino 风格的 API 用于操作 GPIO，如设置 GPIO 模式和输出电平、读取 GPIO 输入电平、配置 GPIO 外部中断。本文说明了如何使用 RT-Thread 的通用 GPIO 设备驱动。

### 1.2 本文的结构

本文首先描述了 RT-Thread 通用 GPIO 设备驱动的基本情况，接下来给出了在正点原子 STM32F4 探索者开发板上验证的代码示例，最后详细描述了通用 GPIO 设备驱动 API 的参数取值和注意事项。

## 2 问题阐述

RT-Thread 提供了一套简单的 I/O 设备管理框架，它把 I/O 设备分成了三层进行处理：应用层、I/O 设备管理层、硬件驱动层。应用程序通过 RT-Thread 的设备操作接口获得正确的设备驱动，然后通过这个设备驱动与底层 I/O 硬件设备进行数据（或控制）交互。RT-Thread 提供给上层应用的是一个抽象的设备操作接口，给下层设备提供的是底层驱动框架。对于通用 GPIO 设备，应用程序既可以通过设备操作接口访问，又可以直接通过通用 GPIO 设备驱动来访问。一般来说，我们都是使用第二种方式，那么如何在 RT-Thread 中使用通用 GPIO 设备驱动从而操作 GPIO 呢？



图 1: RT-Thread 设备管理框架

### 3 问题的解决

本文基于正点原子 STM32F4 探索者开发板，给出了通用 GPIO 设备的具体应用示例代码，包含管脚输入、输出和外部中断的使用方法。由于 RT-Thread 上层应用 API 的通用性，因此这些代码不局限于具体的硬件平台，用户可以轻松将它移植到其它平台上。正点原子 STM32F4 探索者开发板使用的 MCU 是 STM32F407ZGT6，板载 2 颗 LED 和 4 个独立按键。LED 分别连接到 MCU 的 GPIOF9、GPIOF10，KEY0 按键连接到 GPIOE4，KEY1 按键连接到 GPIOE3，KEY2 按键连接到 GPIOE2，WK\_UP 按键连接到 GPIOA0，2 颗 LED 均为低电平点亮，独立按键 KEY0、KEY1、KEY2 按下为低电平；WK\_UP 按下为高电平。

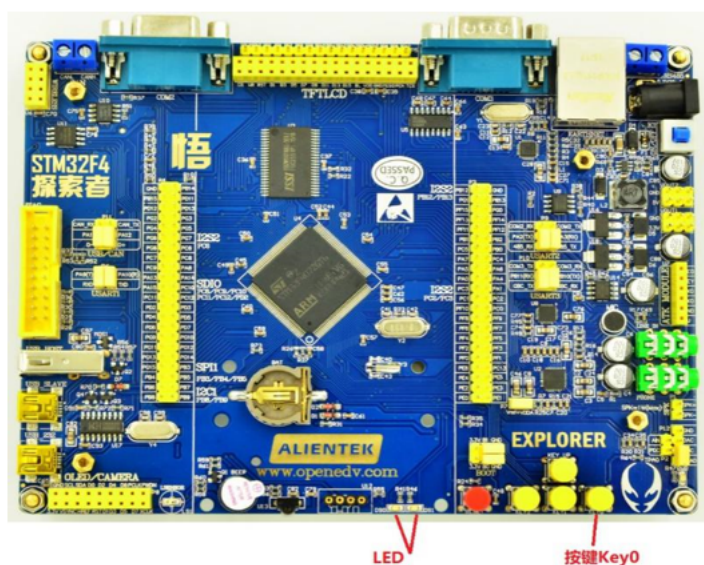


图 2: 正点原子开发板

### 3.1 准备和配置工程

1. 下载 [RT-Thread 源码](#)
  2. 下载 [GPIO 设备驱动示例代码](#)
  3. 进入 `rt-thread\bsp\stm32f4xx-HAL` 目录，在 `env` 命令行中输入 `menuconfig`，进入配置界面，使用 `menuconfig` 工具（学习如何使用）配置工程。
- (1) 在 `menuconfig` 配置界面依次选择 `RT-Thread Components` —> `Device Drivers` —> `Using generic GPIO device drivers`，如图所示：

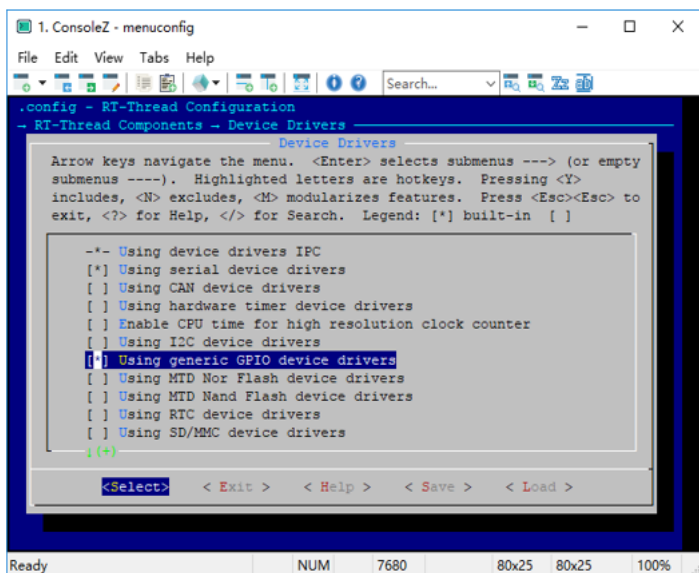


图 3: menuconfig 中开启 GPIO 驱动

- (2) 输入命令 `scons --target=mdk5 -s` 生成 `mdk5` 工程。将示例代码附带的主 `main.c` 替换掉 `bsp` 中的 `main.c`，如图所示：

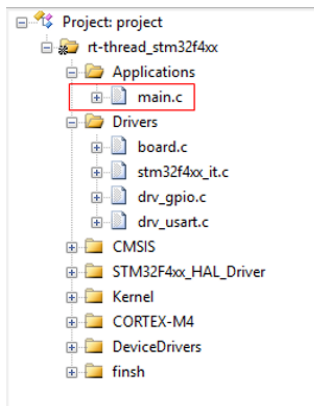
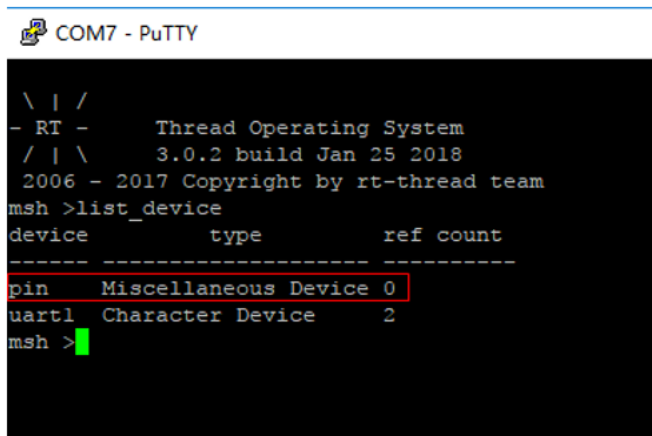


图 4: 加入测试代码

- (3) 编译，下载程序，在终端输入 `list_device` 命令可以看到 device 是 pin、类型是 Miscellaneous Device 就说明通用 GPIO 设备驱动添加成功了。



```
\ | /
- RT -      Thread Operating System
/ | \      3.0.2 build Jan 25 2018
2006 - 2017 Copyright by rt-thread team

msh >list_device
device          type          ref count
-----
pin      Miscellaneous Device  0
uart1    Character Device      2
msh >
```

图 5: 查看 pin 设备

下面是 3 个通用 GPIO 设备驱动 API 应用示例，分别是：GPIO 输出、GPIO 输入、GPIO 外部中断，这些代码在正点原子 STM32F4 探索者开发板上验证通过。

## 3.2 GPIO 输出配置

示例 1: 配置 GPIO 为输出，点亮 LED。根据原理图，GPIOF9 连接到了板载红色 LED，丝印为 DS0；GPIOF10 连接到了板载绿色 LED，丝印为 DS1。GPIOF9 输出低电平则点亮 DS0，GPIOF9 输出高电平则 DS0 不亮；GPIOF10 输出低电平则点亮 DS1，GPIOF10 输出高电平则 DS1 不亮。

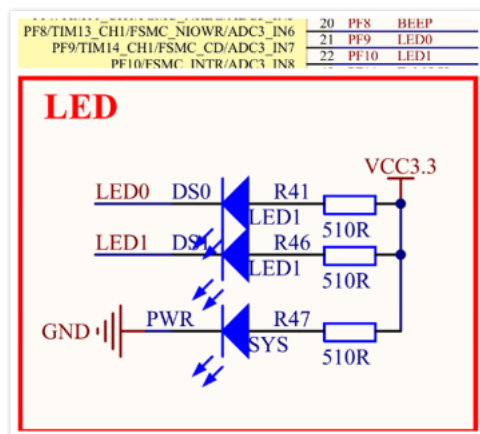


图 6: LED 原理图

```

#define LED0 21 //PF9--21, 在 drv_gpio.c 文件 pin_index pins[] 中查到 PF9
    编号为 21
#define LED1 22 //PF10--22, 在 drv_gpio.c 文件 pin_index pins[] 中查到
    PF10 编号为 22
void led_thread_entry(void* parameter)
{
    // 设置管脚为输出模式
    rt_pin_mode(LED0, PIN_MODE_OUTPUT);
    // 设置管脚为输出模式
    rt_pin_mode(LED1, PIN_MODE_OUTPUT);
    while (1)
    {
        // 输出低电平, LED0 亮
        rt_pin_write(LED0, PIN_LOW);
        // 输出低电平, LED1 亮
        rt_pin_write(LED1, PIN_LOW);
        // 挂起 500ms
        rt_thread_delay(rt_tick_from_millisecond(500));

        // 输出高电平, LED0 灭
        rt_pin_write(LED0, PIN_HIGH);
        // 输出高电平, LED1 灭
        rt_pin_write(LED1, PIN_HIGH);
        // 挂起 500ms
        rt_thread_delay(rt_tick_from_millisecond(500));
    }
}

```

在线程入口函数 led\_thread\_entry 里首先调用 rt\_pin\_mode 设置管脚模式为输出模式, 然后就进入 while(1) 循环, 间隔 500ms 调用 rt\_pin\_write 来改变 GPIO 输出电平。下面是创建线程的代码:

```

rt_thread_t tid; // 线程句柄
/* 创建 led 线程 */
tid = rt_thread_create("led",
                        led_thread_entry,
                        RT_NULL,
                        1024,
                        3,
                        10);
/* 创建成功则启动线程 */
if (tid != RT_NULL)
    rt_thread_startup(tid);

```

编译、下载程序，我们将看到 LED 间隔 500ms 闪烁的现象。

### 3.3 GPIO 输入配置

示例 2：配置 GPIOE3、GPIOE2 为上拉输入，GPIOA0 为下拉输入，检测按键信号。根据原理图，GPIOE3 连接到按键 KEY1，按键被按下时 GPIOE3 应读取到低电平，按键没有被按下时 GPIOE3 应读取到高电平；GPIOE2 连接到按键 KEY2，按键被按下时 GPIOE2 应读取到低电平，按键没有被按下时 GPIOE2 应读取到高电平；GPIOA0 连接到按键 WK\_UP，按键被按下时 GPIOA0 应读取到高电平，按键没有被按下时 GPIOA0 应读取到低电平。

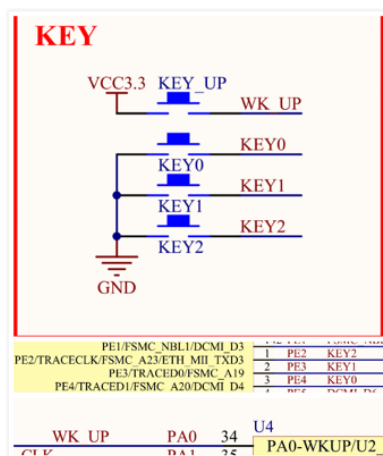


图 7: 按键原理图

```
#define KEY1    2    //PE3--2, 在 drv_gpio.c 文件 pin_index pins[] 中查到
                    //PE3 编号为 2
#define KEY2    1    //PE2--1, 在 drv_gpio.c 文件 pin_index pins[] 中查到
                    //PE2 编号为 1
#define WK_UP   34   //PA0--34, 在 drv_gpio.c 文件 pin_index pins[] 中查到
                    //PA0 编号为 34

void key_thread_entry(void* parameter)
{
    //PE2、PE3 设置上拉输入
    rt_pin_mode(KEY1, PIN_MODE_INPUT_PULLUP);
    rt_pin_mode(KEY2, PIN_MODE_INPUT_PULLUP);
    //PA0 设置为下拉输入
    rt_pin_mode(WK_UP, PIN_MODE_INPUT_PULLDOWN);

    while (1)
    {
        // 检测到低电平，即按键 1 按下了
    }
}
```



```

        if (rt_pin_read(KEY1) == PIN_LOW)
        {
            rt_kprintf("key1 pressed!\n");
        }
        // 检测到低电平，即按键 2 按下了
        if (rt_pin_read(KEY2) == PIN_LOW)
        {
            rt_kprintf("key2 pressed!\n");
        }
        // 检测到高电平，即按键 wp 按下了
        if (rt_pin_read(WK_UP) == PIN_HIGH)
        {
            rt_kprintf("WK_UP pressed!\n");
        }
        // 挂起 10ms
        rt_thread_delay(rt_tick_from_millisecond(10));
    }
}

```

在线程入口函数 `key_thread_entry` 里首先调用 `rt_pin_mode` 设置管脚 GPIOE3 为上拉输入模式。这样当用户按下按键 KEY1 时，GPIOE3 读取到的电平是低电平；按键未被按下时，GPIOE3 读取到的电平是高电平。然后进入 `while(1)` 循环，调用 `rt_pin_read` 读取管脚 GPIOE3 电平，如果读取到低电平则表示按键 KEY1 被按下，就在终端打印字符串“key1 pressed!”。每隔 10ms 检测一次按键输入情况。下面是创建线程的代码：

```

rt_thread_t tid;
/* 创建 key 线程 */
tid = rt_thread_create("key",
                        key_thread_entry,
                        RT_NULL,
                        1024,
                        2,
                        10);
/* 创建成功则启动线程 */
if (tid != RT_NULL)
    rt_thread_startup(tid);

```

编译、下载程序，我们按下开发板上的用户按键，终端将打印提示字符。

### 3.4 GPIO 中断配置

示例 3: 配置 GPIO 为外部中断模式、下降沿触发，检测按键信号。根据原理图，GPIOE4 连接到按键 KEY0，按键被按下时 MCU 应探测到电平下降沿。

```

#define KEY0    3    //PE4--3, 在 gpio.c 文件 pin_index pins[] 中查到 PE4
                      编号为 3
void hdr_callback(void *args)// 回调函数
{
    char *a = args;// 获取参数
    rt_kprintf("key0 down! %s\n",a);
}

void irq_thread_entry(void* parameter)
{
    // 上拉输入
    rt_pin_mode(KEY0, PIN_MODE_INPUT_PULLUP);
    // 绑定中断, 下降沿模式, 回调函数名为 hdr_callback
    rt_pin_attach_irq(KEY0, PIN_IRQ_MODE_FALLING, hdr_callback, (void*)"
        callback
args");
    // 使能中断
    rt_pin_irq_enable(KEY0, PIN_IRQ_ENABLE);
}

```

在线程入口函数 `irq_thread_entry` 里首先调用 `rt_pin_attach_irq` 设置管脚 GPIOE4 为下降沿中断模式, 并绑定了中断回调函数, 还传入了字符串 “callback args”。然后调用 `rt_pin_irq_enable` 使能中断, 这样按键 KEY0 被按下时 MCU 会检测到电平下降沿, 触发外部中断, 在中断服务程序中会调用回调函数 `hdr_callback`, 在回调函数中打印传入的参数和提示信息。下面是创建线程的代码:

```

rt_thread_t tid;// 线程句柄
/* 创建 irq 线程 */
tid = rt_thread_create("exirq",
    irq_thread_entry,
    RT_NULL,
    1024,
    4,
    10);
/* 创建成功则启动线程 */
if (tid != RT_NULL)
    rt_thread_startup(tid);

```

编译、下载程序, 我们按下按键 KEY0, 终端将打印提示字符。

### 3.5 I/O 设备管理框架和通用 GPIO 设备的联系

RT-Thread 自动初始化功能依次调用 `rt_hw_pin_init` ==> `rt_device_pin_register` ==> `rt_device_register` 完成了 GPIO 硬件初始化。`rt_device_register` 注册设备类型为 `RT_Device_Class_Miscellaneous`，即杂类设备，从而我们就可以使用统一的 API 操作 GPIO。

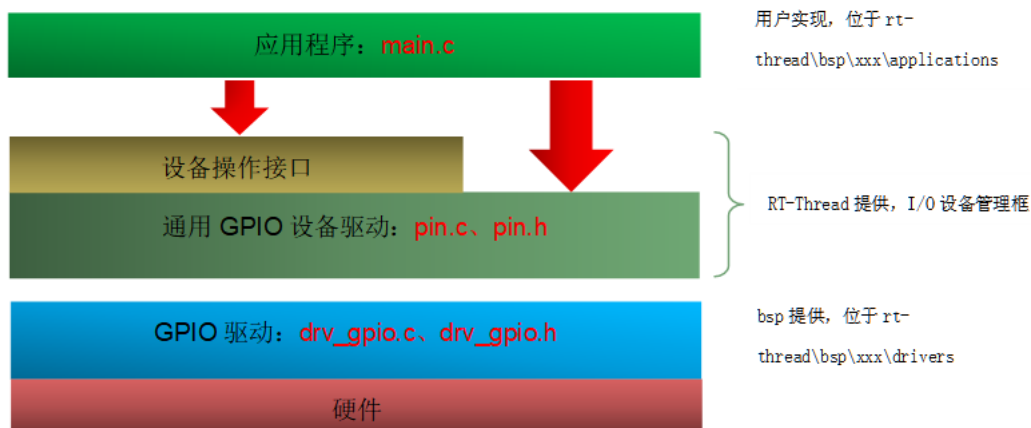


图 8: 通用 GPIO 驱动和设备管理框架联系

更多关于 I/O 设备管理框架的说明和串口驱动实现细节，请参考《RT-Thread 编程手册》第 6 章 I/O 设备管理

在线查看地址: [链接](#)

## 4 参考

### 4.1 本文所有相关的 API

用户应用代码要使用 RT-Thread GPIO 驱动接口需在 `menuconfig` 中开启 GPIO 驱动，引用头文件 `rtdevice.h`。

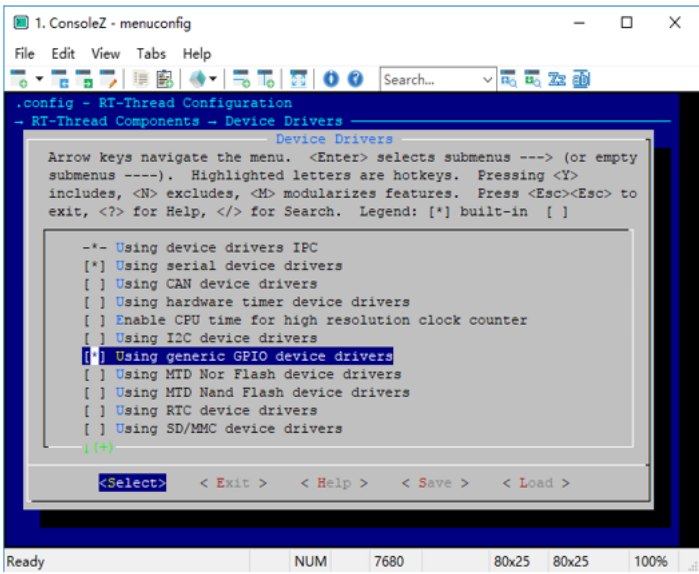


图 9: menuconfig 中开启 GPIO 驱动

4.1.1. API 列表

API	头文件
rt_pin_mode	rt-thread\components\drivers\include\drivers\pin.h
rt_pin_write	rt-thread\components\drivers\include\drivers\pin.h
rt_pin_read	rt-thread\components\drivers\include\drivers\pin.h
rt_pin_attach_irq	rt-thread\components\drivers\include\drivers\pin.h
rt_pin_detach_irq	rt-thread\components\drivers\include\drivers\pin.h
rt_pin_irq_enable	rt-thread\components\drivers\include\drivers\pin.h

4.1.2. 核心 API 详解

4.1.2.1. rt\_pin\_mode() 函数原型

```
void rt_pin_mode(rt_base_t pin, rt_base_t mode)
```

函数参数

参数	描述
pin	管脚编号
mode	模式

函数返回无

此函数可设置管脚模式。

管脚编号，由驱动定义，在 `drv_gpio.c` 的 `pin_index pins[]` 中可以找到，以本文使用的 STM32F407ZGT6 为例，该芯片管脚数为 100，在 `pin_index pins[]` 中可以找到如下代码：

```
352 #endif
353 #if (STM32F4xx_PIN_NUMBERS == 100 && !defined(STM32F469xx) && !defined(STM32F479xx))
354     __STM32_PIN_DEFAULT,
355     __STM32_PIN(1, E, 2),
356     __STM32_PIN(2, E, 3),
357     __STM32_PIN(3, E, 4),
358     __STM32_PIN(4, E, 5),
359     __STM32_PIN(5, E, 6),
360     __STM32_PIN_DEFAULT,
361     __STM32_PIN(7, C, 13),
362     __STM32_PIN(8, C, 14),
363     __STM32_PIN(9, C, 15),
364     __STM32_PIN_DEFAULT,
365     __STM32_PIN_DEFAULT,
366     __STM32_PIN_DEFAULT,
367     __STM32_PIN_DEFAULT,
368     __STM32_PIN_DEFAULT,
369     __STM32_PIN(15, C, 0),
370     __STM32_PIN(16, C, 1),
371     __STM32_PIN(17, C, 2),
372     __STM32_PIN(18, C, 3),
373     __STM32_PIN_DEFAULT,
374     __STM32_PIN_DEFAULT,
375     __STM32_PIN_DEFAULT,
376     __STM32_PIN_DEFAULT,
```

图 10: pin 编号

其中 `STM32_PIN(1, E, 2)` 表示 GPIOE2 的编号为 1，`STM32_PIN(9, C, 15)` 表示 GPIOC15 的编号为 9，以此类推。`STM32_PIN()` 的第一个参数为管脚编号，第二个参数为端口，第三个参数为管脚号。

模式可取如下 5 种之一：

`PIN_MODE_OUTPUT` 输出，具体模式看 `drv_gpio.c` 源码实现，本文使用的是推挽输出 `PIN_MODE_INPUT` 输入 `PIN_MODE_INPUT_PULLUP` 上拉输入 `PIN_MODE_INPUT_PULLDOWN` 下拉输入 `PIN_MODE_OUTPUT_OD` 开漏输出

4.1.2.2. `rt_pin_write()` 函数原型

```
void rt_pin_write(rt_base_t pin, rt_base_t value)
```

#### 函数参数

参数	描述
pin	管脚编号
value	电平逻辑值，可取 2 种值之一，PIN_LOW 低电平，PIN_HIGH 高电平

#### 函数返回无

此函数可设置管脚输出电平。

#### 4.1.2.3. rt\_pin\_read() 函数原型

```
int rt_pin_read(rt_base_t pin)
```

#### 函数参数

参数	描述
pin	管脚编号

#### 函数返回

返回值	描述
PIN_LOW	低电平
PIN_HIGH	高电平

此函数可读取输入管脚电平。

#### 4.1.2.4. rt\_pin\_attach\_irq() 函数原型

```
rt_err_t rt_pin_attach_irq( rt_int32_t pin, rt_uint32_t mode,
                             void (*hdr)(void *args), void *args)
```

#### 函数参数

参数	描述
pin	管脚编号
mode	中断触发模式
hdr	中断回调函数，用户需要自行定义这个函数，其返回值为 void
args	中断回调函数的参数，不需要时设置为 RT_NULL

#### 函数返回

返回值	描述
RT_EOK	成功
RT_ENOSYS	无系统
RT_EBUSY	忙

中断触发模式可取以下 3 种值之一：

PIN\_IRQ\_MODE\_RISING 上升沿触发 PIN\_IRQ\_MODE\_FALLING 下降沿触发  
PIN\_IRQ\_MODE\_RISING\_FALLING 边沿触发（上升沿和下降沿都触发）

此函数可绑定中断回调函数。

绑定中断回调函数传递字符串示例：

```
rt_pin_attach_irq(3, PIN_IRQ_MODE_FALLING, hdr_callback, (void*)"callback
args");
void hdr_callback(void *args)
{
    char *a = args;

    rt_kprintf("%s", a);
}
```

输出为 “callback args”。

传递数值示例：

```
rt_pin_attach_irq(3, PIN_IRQ_MODE_FALLING, hdr_callback, (void*)6);
void hdr_callback(void *args)
```

```
{
    Int a = (int)args;

    rt_kprintf("%d",a);
}
```

输出为 6。

4.1.2.5. rt\_pin\_detach\_irq() 函数原型

```
rt_err_t rt_pin_detach_irq(rt_int32_t pin)
```

函数参数

参数	描述
pin	管脚编号

函数返回

返回值	描述
RT_EOK	成功
RT_ENOSYS	出错

此函数可使管脚中断回调函数脱离。

4.1.2.6. rt\_pin\_irq\_enable() 函数原型

```
rt_err_t rt_pin_irq_enable(rt_base_t pin, rt_uint32_t enabled)
```

函数参数

参数	描述
pin	管脚编号
enabled	状态，可取 2 种值之一：PIN_IRQ_ENABLE 开启，PIN_IRQ_DISABLE 关闭



## 函数返回

返回值	描述
RT_EOK	成功
RT_ENOSYS	出错