HAL_V2 IRRX

开发指南

版本号: 1.0

发布日期: 2025.07.25

版本历史

版本号	日期	制/修订人	内容描述
1.0	2025.07.25	AWA2215	初始版本

目　录

1 前言

1.1 文档简介

介绍 HAL_V2 中 IRRX 驱动的接口及使用方法，为 GPIO 使用者提供参考。

1.2 目标读者

IRRX 驱动的开发/维护人员。

1.3 适用范围

表 1-1: 适用产品列表

产品名称	内核版本	驱动文件
T536	FreeRTOS	hal_irrx.c
T153	FreeRTOS	hal_irrx.c

1.4 文档约定

1 标志说明

⚠ 注意

- **提醒操作中应注意的事项。不当的操作可能会损坏器件，影响可靠性、降低性能等。**

说明

为准确理解文中指令、正确实施操作而提供的补充或强调信息。

技巧

一些容易忽视的小功能、技巧。了解这些功能或技巧能帮助解决特定问题或者节省操作时间。

1.5 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台
IR	红外模块
RX	红外接收

2 模块介绍

2.1 模块配置介绍

menuconfig 配置项

```
Drivers V2 Config  --->

    [*] enable irrx driver
    [ ]  enable irrx hal APIs test command
```

Driver --->

2.2 模块源码结构

1. 驱动源码

```
hal_v2/hal/source/irrx/
├──── common_irrx.h
├──── hal_irrx.c

├──── Makefile
├──── platform
│    ├──── irrx_sun55iw6.h
│    └──── irrx_sun8iw22.h
└──── platform_irrx.h
```

Kconfig

2. 驱动 APIs 声明头文件

```
hal_v2/include/hal/
└──── hal_irrx.h
```

动 APIs 测试代码

```
hal_v2/hal/examples/irrx/
├──── irrx_receiver_test
│    ├──── irrx_recevier_test.c
│    └──── Makefile
└──── Makefile
```

3 模块接口说明

3.1 对外提供的 API 接口

IR 提供的接口列表, 其他的接口可以看 hal_irrx.h，部分重要接口如下：

1 hal_irrx_init

- 原型: irrx_status_t hal_irrx_init(irrx_port_t port, irrx_init_t *irrx_initstruct)
- 作用：irrx 模块初始化
- 参数：
  - port:irrx 端口
  - irrx_initstruct: 初始化结构体指针
- 返回：

0: 成功
负数: 失败

3.1.2 hal_irrx_deinit

- 原型: void hal_irrx_deinit(irrx_port_t port)
- 作用：irrx 模块去初始化。
- 参数：
  - port:SPI 端口号

:

  - void

3.1.3 hal_irrx_read_fifo_raw_data

- 原型: s32 hal_irrx_read_fifo_raw_data(irrx_port_t port, struct ir_raw_buffer_t *raw_buf, u32 count)
- 作用：读取接收到的 raw 数据

- 参数：
  - port:irrx 端口
  - raw_buf: 存储接收到的 raw 数据
  - count: 要接收的数据大小
- 返回：
  - 0: 成功
  - 负数: 失败

3.1.4 hal_irrx_decode_raw_data

- 原型: u32 hal_irrx_decode_raw_data(u8 *buffer, u32 count)
- 作用：解码接收到的 raw 数据。
- 参数：
  - buffer:raw 数据 buffer
  - count: 数据大小
- 返回：
  - u32：解码后的数据

4 模块使用范例

可参考驱动 APIs 测试代码（hal_v2/hal/examples/irrx/）：

```
#include <hal_irrx.h>
#include <hal_interrupt.h>
#include <hal_gpio.h>
#include <hal_clk.h>
#include <hal_reset.h>

static irrx_port_t port = IRRX_MASTER_0;
static u32 ir_valid_value;
static ir_raw_buffer_t raw_buffer;
static u32 ir_valid_command;
static u32 ir_valid_addr;

int irrx_clk_init()
{
    int ret;
    hal_rst_t rst;
    rst_number_t rst_num;
    hal_clk_t clk_bus, clk, parent_clk;
    clk_number_t clk_bus_num, clk_num, parent_clk_num;

#if defined(CONFIG_ARCH_SUN55IW6)
num = MAKE_RSTn(AW_SYS_CCU, RST_BUS_IRRX0);
    clk_bus_num = MAKE_CLKn(AW_SYS_CCU, CLK_BUS_IRRX0);

    /* clock source */
    clk_num = MAKE_CLKn(AW_SYS_CCU, CLK_IRRX0);
    parent_clk_num = MAKE_CLKn(AW_SRC_CCU, CLK_SRC_HOSC);
#else
    rst_num = MAKE_RSTn(AW_SYS_CCU, RST_BUS_IR_RX0);
    clk_bus_num = MAKE_CLKn(AW_SYS_CCU, CLK_IR_RX0_BUS);

    /* clock source */
    clk_num = MAKE_CLKn(AW_SYS_CCU, CLK_IR_RX0);
    parent_clk_num = MAKE_CLKn(AW_SRC_CCU, CLK_SRC_HOSC);
#endif

    ret = hal_rst_get_by_rst_num(rst_num, &rst);
    if (ret) {
        printf("get reset handle failed, rst_num: %u, ret: %d\n", rst_num, ret);
        return -1;
    }

    ret = hal_n_clk_get_by_clk_num(clk_bus_num, &clk_bus);
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", clk_bus_num, ret);
        return -1;
    }

    ret = hal_n_clk_get_by_clk_num(clk_num, &clk);
```

```
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", clk_num, ret);
        printf("%d\n", __LINE__);
        return -1;
    }

    ret = hal_n_clk_get_by_clk_num(parent_clk_num, &parent_clk);
    if (ret) {
        printf("get clk(%u) failed, ret: %d\n", parent_clk_num, ret);
        printf("%d\n", __LINE__);
        return -1;
    }

    ret = hal_rst_deassert(rst);
    if (ret) {
        printf("deassert reset failed, rst_num: %u, ret: %d\n", rst_num, ret);
        ret = -2;
to exit_with_put_rst;



    ret = hal_n_clk_enable(clk_bus);
    if (ret) {
        printf("enable clk(%u) failed, ret: %d\n", clk_bus_num, ret);
        ret = -2;
        goto exit_with_put_clk;
    }

    ret = hal_n_clk_set_parent(clk, parent_clk);
    if (ret) {
        printf("set the parent of clk(%u) to clk(%u) failed, ret: %d\n", clk_num, parent_clk_num, ret);
        ret = -3;
        goto exit_with_put_parent_clk;
    }

    ret = hal_n_clk_enable(clk);
    if (ret) {
        printf("enable clk(%u) failed, ret: %d\n", clk_num, ret);
        ret = -2;
        goto exit_with_put_clk;
    }

    ret = 0;

exit_with_put_parent_clk:
    hal_n_clk_put(parent_clk);

exit_with_put_clk:
    hal_n_clk_put(clk);

exit_with_put_rst:
    hal_rst_put(rst);
    return ret;
}

static u32 ir_code_is_valid(u32 code)
{
    u32 tmp1, tmp2;

    tmp1 = code & 0x00ff0000;
    tmp2 = (code & 0xff000000) >> 8;
```

```
    if (tmp1 | tmp2) {
        ir_valid_command = tmp1 >> 16;
        ir_valid_addr = code & 0xffff;
        printf("ir_valid_command:0x%x, ir_valid_addr:0x%x\n", ir_valid_command, ir_valid_addr);
        return TRUE;
    } else {
        ir_valid_command = 0;
        ir_valid_addr = 0;
        return FALSE;
    }
}

static hal_irqreturn_t irrx_irq_handler(void *data)
{
    u32 count;
    u32 status;

    /* read out interrupt pending status register */
    status = readl(IRRX_BASE(port) + IR_RXINTS_REG);
    writel(status & 0xff, IRRX_BASE(port) + IR_RXINTS_REG);

    count = (status >> 8) & 0x7f;
    /* rx fifo available interrupt */
    if (hal_irrx_read_fifo_raw_data(port, &raw_buffer, count) != OK) {
        raw_buffer.count = 0;
        return FALSE;
    }

    if (!(status & 0xff)) {
        raw_buffer.count = 0;
        return FALSE;
    }

    /* invalid ir rx interrupt detect */
    if (status & IR_RXINTS_RXPE) {
        /* decode raw data */
        ir_valid_value = hal_irrx_decode_raw_data(raw_buffer.data, raw_buffer.count);

        raw_buffer.count = 0;

        if (ir_valid_value == IR_ERROR_CODE) {
            return FALSE;
        }

        if (ir_code_is_valid(ir_valid_value)) {
            return TRUE;
        }
    }

    /* invalid ir rx interrupt detect */
    if (status & IR_RXINTS_RXOF) {
        /* ir rx fifo full interrupt */
        raw_buffer.count = 0;
        return FALSE;
    }

    return TRUE;
}
```

```
void irrx_recevier_test(void)
{

   printf("--------------IRRX TEST--------------\n");

   irrx_clk_init();

   gpio_init_t gpio_initstruct;
   gpio_initstruct.port = SUNXI_GPIO_J;/* PB4: IR_RX0 */
   gpio_initstruct.port_num = PIN_12;
   gpio_initstruct.mul_sel = GPJ_IR0_RX;
   hal_gpio_init(&gpio_initstruct);

   irrx_init_t irrx_initstruct;
   irrx_initstruct.clk_div = IRRX_CLK_DIV256;
   irrx_initstruct.mode = IRRX_BOTH_PULSE;
   hal_irrx_init(port, &irrx_initstruct);

   hal_irq_init_t intc_initstruct;
   intc_initstruct.irqn = IRRX0_CPUX_IRQ;
   intc_initstruct.preemptionpriority = 0;
   intc_initstruct.subpriority = 0;
   intc_initstruct.cmd = true;
   intc_initstruct.isr_info.func = irrx_irq_handler;
   intc_initstruct.isr_info.parg = NULL;
   hal_v2_intc_irq_config(&intc_initstruct);

   /* Please press the IR remote control and wait for the interruption to come */
}
```

5 FAQ

无

声明

商标声明

免责声明