Linux IR-TX
开发指南

版本号: 1.3
发布日期: 2024.11.26

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.11.1	XAA0248	初始版本
1.1	2023.3.4	XAA0311	根据驱动的变化，修改 dts
1.2	2023.11.27	XAA0311	添加 hexdump 的举例
1.3	2024.11.26	AWA2215	增加 Linux-6.6 内核版本支持，调整文档结构

目　录

1 前言

1.1 文档简介

介绍 Sunxi 平台上 IR-TX 驱动接口与调试方法，为 IR-TX 模块开发提供参考。

1.2 目标读者

IR-TX 模块内核层以及应用层的开发、维护人员。

1.3 适用范围

表 1-1: 适用产品列表

内核版本	驱动文件
Linux-5.15	${SDK}/bsp/drivers/ir-tx/sunxi-ir-tx.c
Linux-6.6	${SDK}/bsp/drivers/ir-tx/sunxi-ir-tx.c

1.4 文档约定

1.4.1 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。

1.4.2 地址与数据描述方法约定

本文档在描述地址、数据时遵循如下约定：

表 1-2: 地址与数据描述方法约定

符号	例子	说明
0x	0x0200，0x79	地址或数据以 16 进制表示。
0b	0b010，0b00 000 111	数据采用二进制表示 (寄存器描述除外)。
X	00X，XX1	数据描述中，X 代表 0 或 1。 例如，00X 代表 000 或 001；XX1 代表 001，011，101 或 111。

1.4.3 数值单位约定

本文档在描述数据容量（如 NAND 容量）时，单位词头代表的是 1024 的倍数；描述频率、数据速率等时则代表的是 1000 的倍数。具体如下：

表 1-3: 数值单位约定

类型	符号	对应数值
数据容量（如 NAND 容量）	1 K	1024
	1 M	1 048 576
	1 G	1 073 741 824
频率，数据速率等	1 k	1000
	1 M	1 000 000
	1 G	1 000 000 000

1.5 相关术语介绍

1.5.1 硬件术语

表 1-4: 硬件术语

术语	解释说明
IR	Infrared Remote，红外模块

1.5.2 软件术语

表 1-5: 软件术语

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台。
TX	发送。
NEC 协议	一种红外标准传输协议。

2 模块介绍

2.1 模块功能介绍

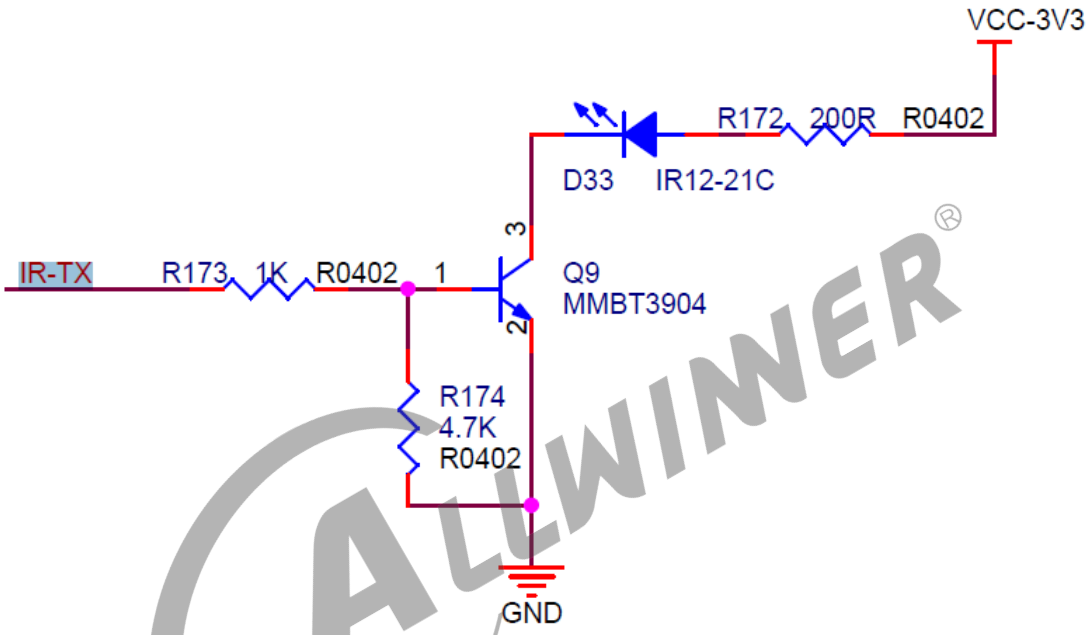

图 2-1: IR-TX 模块

Sunxi IR-TX 为红外发送模块，可输出红外波形，常用载波频率为 38KHz。它可以将数据转变成一定长度的高低电平序列，每个有效脉冲的数据都按照游程编码的方式，以字节为单位被缓存到 TX FIFO 中。每个 byte 的 bit7 代表要发送波形的极性，即 1 代表高电平，0 代表低电平，bit[6:0] 代表这个波形的长度，单位是 Reference CLK 的一个周期（Ts=Fclk/RCS，Fclk 为 Sunxi IR-TX 模块的时钟源，RCS 为参考时钟分频系数，由 TCR 寄存器（0x08）bit[3:1] 设定）。

2.2 模块配置介绍

在不同的 Sunxi 硬件平台中，IR-TX 控制器的数目不同；但对于同一块板子上的每一个 IR-TX 控制器来说, 模块配置类似，本小节展示 Sunxi 平台上的 ir-tx 控制器配置（其他 IR-TX 控制器配置类似）。

2.2.1 device tree 默认配置

设备树中存在的是该类芯片所有平台的模块配置，IR-TX 的设备树配置如下所示：

```
irtx: ir@2003000 {
    compatible = "allwinner,irtx";
    reg = <0x0 0x02003000 0x0 0x400>;
    interrupts = <GIC_SPI 26 IRQ_TYPE_LEVEL_HIGH>;
    clocks = <&ccu CLK_BUS_IRTX>, <&dcxo24M>, <&ccu CLK_IRTX>;
    clock-names = "bus", "pclk", "mclk";
    resets = <&ccu RST_BUS_IRTX>;
    status = "disabled";
};
```

2.2.2 board.dts 板级配置

board.dts 用于保存每一个板级平台的设备信息（如 demo 板，perf1 板，ver1 板等等），里面的配置信息会覆盖上面的 device tree 默认配置信息。

对应 board.dts 里面 ir-tx 的具体配置如下：

```
&pio {
    irtx_pins_default: irtx@0 {
        pins = "PH18";
        function = "cir";
    };

    irtx_pins_sleep: irtx@1 {
        pins = "PH18";
        function = "gpio_in";
    };
};

&irtx {
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&irtx_pins_default>;
    pinctrl-1 = <&irtx_pins_sleep>;
    status = "disabled";
};
```

2.2.3 kernel menuconfig 配置

在 SDK 根目录中执行./build.sh menuconfig，选择 **Allwinner BSP** 选项进入下一级配置 **Device Drivers**，接着进入 **IR-TX Drivers** 选项，如下图所示：

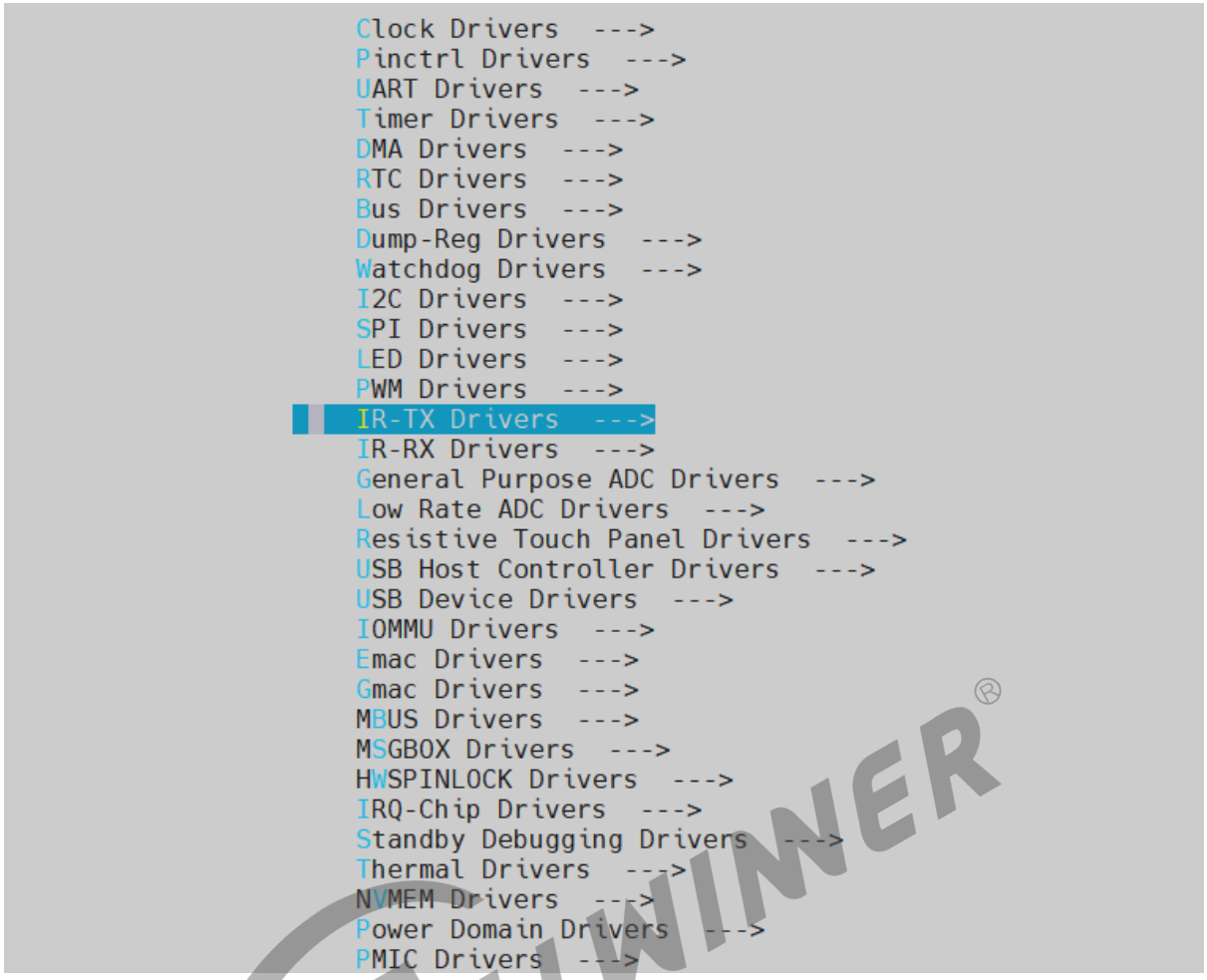

图 2-2: IR-TX Drivers

选择 **IR-TX Support for Allwinner SoCs** 选项，可选择直接编译进内核，也可编译成模块。如下图所示：

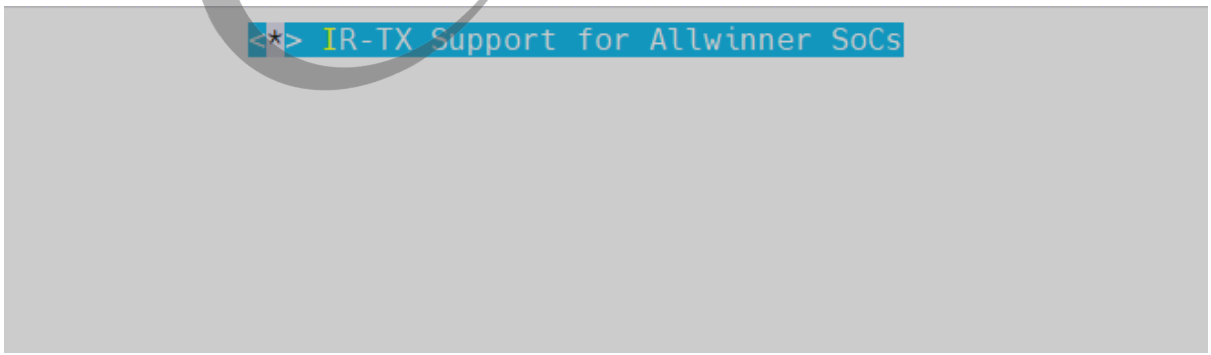

图 2-3: IR-TX Support for Allwinner SoCs

2.3 源码模块结构

源代码位于 sdk/bsp/drivers/ir-tx/目录下：

```
sdk/bsp/drivers/ir-tx/
├──── sunxi-ir-tx.c          // Sunxi平台的ir-tx驱动代码
```

3 接口设计

IR-TX 模块采用通用的 RC 框架进行读写，所以可以使用通用的接口。

3.1 外部接口

IR-TX 模块在 Linux 内核中是作为字符设备使用，所以可以使用相关字符设备接口来对 IR-TX 模块进行相应的读写和配置操作。相关定义在 evdev.c 文件里面。下面介绍几个比较有用的函数：

3.1.1 evdev_open()

- 函数原型：static int evdev_open(struct inode *inode, struct file *file)。
- 功能描述：程序（C 语言等）使用 open(file) 时调用的函数。打开一个 IR-TX 模块设备。
- 参数说明：inode：inode 节点；file：file 结构体。
- 返回值：文件描述符。

3.1.2 evdev_read()

- 函数原型：static ssize_t evdev_read(struct file *file, char __user buffer, size_t count, loff_t ppos)。
- 功能描述：程序（C 语言等）调用 read() 时调用的函数。读取 IR-TX 模块上报事件数据。
- 参数说明：file，file 结构体；buf，写数据 buf；offset, 文件偏移。
- 返回值：成功返回读取的字节数，失败返回负数。

3.1.3 evdev_write()

- 函数原型：static ssize_t evdev_write(struct file *file, const char __user buffer, size_t count, loff_t ppos)。
- 功能描述：程序（C 语言等）调用 write() 时调用的函数。向 IR-TX 模块写入上报事件。
- 参数说明：file，file 结构体；buf，读数据 buf；offset, 文件偏移。
- 返回值：成功返回 0，失败返回负数。

3.1.4 evdev_ioctl()

- 函数原型：static long evdev_ioctl(struct file *file, unsigned int cmd, unsigned long arg)。
- 功能描述：程序（C 语言等）调用 ioctl() 时调用的函数。管理相关的 IR-TX 模块功能。
- 参数说明：file，file 结构体，cmd，指令，arg，其他参数。
- 返回值：成功返回 0，失败返回负数。

找到 IR-TX 模块对应的 eventX(如 dev/input/event0) 文件，通过 IR-TX 的用户层 demo 调用上述的接口。

3.2 EVENT 接口

3.2.1 获取 IR-TX 模块上报数据

在内核中，查看 /proc/bus/input/devices，确认 IR-TX 模块的设备节点，看下面的 Handlers 属性。

```
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="sunxi-ir-tx"
P: Phys=sunxi-ir-tx/input0
S: Sysfs=/devices/platform/soc@3000000/2003000.ir/rc/rc0/ir_tx
U: Uniq=
H: Handlers=kbd event3
B: PROP=20
B: EV=100017
B: KEY=1000000 0 0
B: REL=3
B: MSC=10
```

其中，在读取到 event 节点的数据后，我们可以进行分析这些数据：每行的开头 4 个字节是 hexdump 打印的长度信息，后面跟着的是 16 字节的数据，struct timeval 占了 8 个字节，后面是 2 个字节的 type，2 个字节的 code，4 个字节的 value。具体实现也可以在内核代码中查看 input_event 结构体查看。

android 提供了 getevent 来获取输入设备的信息, 作用跟上面 hexdump /dev/input/event2 类似。具体用法如下：

```
Usage: getevent [-t] [-n] [-s switchmask] [-S] [-v [mask]] [-d] [-p] [-i] [-l] [-q] [-c count] [-r] [device]
  -t: show time stamps   //前部打印时间
  -n: don't print newlines
  -s: print switch states for given bits
  -S: print all switch states
  -v: verbosity mask //打印设备的简易信息，同时也获取上报信息
  -d: show HID descriptor, if available
-p: show possible events (errs, dev, name, pos. events)
  -i: show all device info and possible events   //打印出各个设备的具体信息
  -l: label event types and names in plain text   //打印出上报事件类型
  -q: quiet (clear verbosity mask)   //不打印设备信息，只捕获事件
```

-c: print given number of events then exit
-r: print rate events are received

可以通过 cmd 进入 adb shell 直接输入 getevent -l /dev/input/event3。

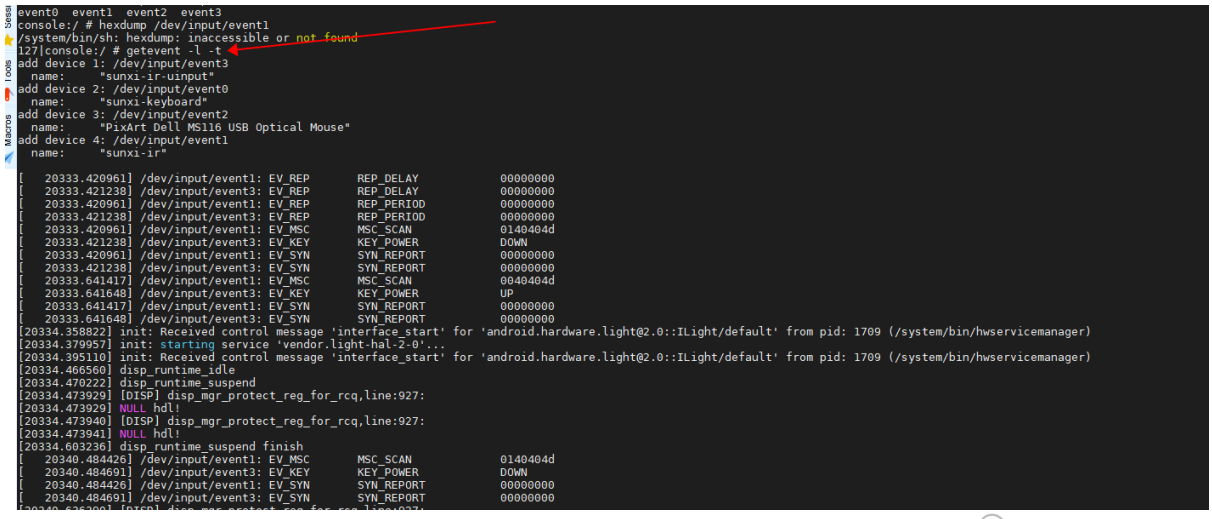

图 3-1

4 模块使用范例

下面是一个用来读取 IR-TX 模块的按键输入的一个 Demo。

ir-tx.h文件

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
/*
 * lirc.h - linux infrared remote control header file
 * last modified 2010/07/13 by Jarod Wilson
 */

#ifndef _APP_IR_H_
#define _APP_IR_H_

#include <linux/types.h>
#include <linux/ioctl.h>

#define PULSE_BIT      0x01000000
#define PULSE_MASK     0x00FFFFFF

#define LIRC_MODE2_SPACE    0x00000000
#define LIRC_MODE2_PULSE    0x01000000
#define LIRC_MODE2_FREQUENCY 0x02000000
#define LIRC_MODE2_TIMEOUT   0x03000000

#define LIRC_VALUE_MASK     0x00FFFFFF
#define LIRC_MODE2_MASK     0xFF000000

#define LIRC_SPACE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_SPACE)
#define LIRC_PULSE(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_PULSE)
#define LIRC_FREQUENCY(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_FREQUENCY)
#define LIRC_TIMEOUT(val) (((val)&LIRC_VALUE_MASK) | LIRC_MODE2_TIMEOUT)

#define LIRC_VALUE(val) ((val)&LIRC_VALUE_MASK)
#define LIRC_MODE2(val) ((val)&LIRC_MODE2_MASK)

#define LIRC_IS_SPACE(val) (LIRC_MODE2(val) == LIRC_MODE2_SPACE)
#define LIRC_IS_PULSE(val) (LIRC_MODE2(val) == LIRC_MODE2_PULSE)
#define LIRC_IS_FREQUENCY(val) (LIRC_MODE2(val) == LIRC_MODE2_FREQUENCY)
#define LIRC_IS_TIMEOUT(val) (LIRC_MODE2(val) == LIRC_MODE2_TIMEOUT)

/* used heavily by lirc userspace */
#define lirc_t int

/*** lirc compatible hardware features ***/

#define LIRC_MODE2SEND(x) (x)
#define LIRC_SEND2MODE(x) (x)
#define LIRC_MODE2REC(x) ((x) << 16)
#define LIRC_REC2MODE(x) ((x) >> 16)

#define LIRC_MODE_RAW              0x00000001
```

```
#define LIRC_MODE_PULSE             0x00000002
#define LIRC_MODE_MODE2             0x00000004
#define LIRC_MODE_SCANCODE          0x00000008
#define LIRC_MODE_LIRCCODE          0x00000010


#define LIRC_CAN_SEND_RAW           LIRC_MODE2SEND(LIRC_MODE_RAW)
#define LIRC_CAN_SEND_PULSE         LIRC_MODE2SEND(LIRC_MODE_PULSE)
#define LIRC_CAN_SEND_MODE2         LIRC_MODE2SEND(LIRC_MODE_MODE2)
#define LIRC_CAN_SEND_LIRCCODE      LIRC_MODE2SEND(LIRC_MODE_LIRCCODE)

#define LIRC_CAN_SEND_MASK          0x0000003f

#define LIRC_CAN_SET_SEND_CARRIER     0x00000100
#define LIRC_CAN_SET_SEND_DUTY_CYCLE  0x00000200
#define LIRC_CAN_SET_TRANSMITTER_MASK 0x00000400

#define LIRC_CAN_REC_RAW            LIRC_MODE2REC(LIRC_MODE_RAW)
#define LIRC_CAN_REC_PULSE          LIRC_MODE2REC(LIRC_MODE_PULSE)
#define LIRC_CAN_REC_MODE2          LIRC_MODE2REC(LIRC_MODE_MODE2)
#define LIRC_CAN_REC_SCANCODE       LIRC_MODE2REC(LIRC_MODE_SCANCODE)
#define LIRC_CAN_REC_LIRCCODE       LIRC_MODE2REC(LIRC_MODE_LIRCCODE)

#define LIRC_CAN_REC_MASK           LIRC_MODE2REC(LIRC_CAN_SEND_MASK)

#define LIRC_CAN_SET_REC_CARRIER      (LIRC_CAN_SET_SEND_CARRIER << 16)
#define LIRC_CAN_SET_REC_DUTY_CYCLE   (LIRC_CAN_SET_SEND_DUTY_CYCLE << 16)

#define LIRC_CAN_SET_REC_DUTY_CYCLE_RANGE 0x40000000
#define LIRC_CAN_SET_REC_CARRIER_RANGE    0x80000000
#define LIRC_CAN_GET_REC_RESOLUTION       0x20000000
#define LIRC_CAN_SET_REC_TIMEOUT          0x10000000
#define LIRC_CAN_SET_REC_FILTER           0x08000000

#define LIRC_CAN_MEASURE_CARRIER          0x02000000
#define LIRC_CAN_USE_WIDEBAND_RECEIVER    0x04000000

#define LIRC_CAN_SEND(x) ((x)&LIRC_CAN_SEND_MASK)
#define LIRC_CAN_REC(x) ((x)&LIRC_CAN_REC_MASK)

#define LIRC_CAN_NOTIFY_DECODE            0x01000000

/*** IOCTL commands for lirc driver ***/

#define LIRC_GET_FEATURES           _IOR('i', 0x00000000, __u32)

#define LIRC_GET_SEND_MODE          _IOR('i', 0x00000001, __u32)
#define LIRC_GET_REC_MODE           _IOR('i', 0x00000002, __u32)
#define LIRC_GET_REC_RESOLUTION     _IOR('i', 0x00000007, __u32)

#define LIRC_GET_MIN_TIMEOUT        _IOR('i', 0x00000008, __u32)
#define LIRC_GET_MAX_TIMEOUT        _IOR('i', 0x00000009, __u32)

/* code length in bits, currently only for LIRC_MODE_LIRCCODE */
#define LIRC_GET_LENGTH             _IOR('i', 0x0000000f, __u32)

#define LIRC_SET_SEND_MODE          _IOW('i', 0x00000011, __u32)
#define LIRC_SET_REC_MODE           _IOW('i', 0x00000012, __u32)
/* Note: these can reset the according pulse_width */
#define LIRC_SET_SEND_CARRIER       _IOW('i', 0x00000013, __u32)
```

```
#define LIRC_SET_REC_CARRIER         _IOW('i', 0x00000014, __u32)
#define LIRC_SET_SEND_DUTY_CYCLE     _IOW('i', 0x00000015, __u32)
#define LIRC_SET_TRANSMITTER_MASK    _IOW('i', 0x00000017, __u32)

/*
 * when a timeout != 0 is set the driver will send a
 * LIRC_MODE2_TIMEOUT data packet, otherwise LIRC_MODE2_TIMEOUT is
 * never sent, timeout is disabled by default
 */
#define LIRC_SET_REC_TIMEOUT         _IOW('i', 0x00000018, __u32)

/* 1 enables, 0 disables timeout reports in MODE2 */
#define LIRC_SET_REC_TIMEOUT_REPORTS  _IOW('i', 0x00000019, __u32)

/*
 * if enabled from the next key press on the driver will send
 * LIRC_MODE2_FREQUENCY packets
 */
#define LIRC_SET_MEASURE_CARRIER_MODE   _IOW('i', 0x0000001d, __u32)

/*
 * to set a range use LIRC_SET_REC_CARRIER_RANGE with the
 * lower bound first and later LIRC_SET_REC_CARRIER with the upper bound
 */
#define LIRC_SET_REC_CARRIER_RANGE    _IOW('i', 0x0000001f, __u32)

#define LIRC_SET_WIDEBAND_RECEIVER    _IOW('i', 0x00000023, __u32)

/*
 * Return the recording timeout, which is either set by
 * the ioctl LIRC_SET_REC_TIMEOUT or by the kernel after setting the protocols.
 */
#define LIRC_GET_REC_TIMEOUT         _IOR('i', 0x00000024, __u32)

/*
 * struct lirc_scancode - decoded scancode with protocol for use with
 *   LIRC_MODE_SCANCODE
 *
 * @timestamp: Timestamp in nanoseconds using CLOCK_MONOTONIC when IR
 *   was decoded.
 * @flags: should be 0 for transmit. When receiving scancodes,
 *   LIRC_SCANCODE_FLAG_TOGGLE or LIRC_SCANCODE_FLAG_REPEAT can be set
 *   depending on the protocol
 * @rc_proto: see enum rc_proto
 * @keycode: the translated keycode. Set to 0 for transmit.
 * @scancode: the scancode received or to be sent
 */
struct lirc_scancode {
  __u64  timestamp;
  __u16  flags;
  __u16  rc_proto;
  __u32  keycode;
  __u64  scancode;
};

/* Set if the toggle bit of rc-5 or rc-6 is enabled */
#define LIRC_SCANCODE_FLAG_TOGGLE  1
/* Set if this is a nec or sanyo repeat */
#define LIRC_SCANCODE_FLAG_REPEAT  2
enum rc_proto {
```

```
    RC_PROTO_UNKNOWN   = 0,
    RC_PROTO_OTHER     = 1,
    RC_PROTO_RC5       = 2,
    RC_PROTO_RC5X_20   = 3,
    RC_PROTO_RC5_SZ    = 4,
    RC_PROTO_JVC       = 5,
    RC_PROTO_SONY12    = 6,
    RC_PROTO_SONY15    = 7,
    RC_PROTO_SONY20    = 8,
    RC_PROTO_NEC       = 9,
    RC_PROTO_NECX      = 10,
    RC_PROTO_NEC32     = 11,
    RC_PROTO_SANYO     = 12,
    RC_PROTO_MCIR2_KBD = 13,
    RC_PROTO_MCIR2_MSE = 14,
    RC_PROTO_RC6_0     = 15,
    RC_PROTO_RC6_6A_20 = 16,
    RC_PROTO_RC6_6A_24 = 17,
    RC_PROTO_RC6_6A_32 = 18,
    RC_PROTO_RC6_MCE   = 19,
    RC_PROTO_SHARP     = 20,
    RC_PROTO_XMP       = 21,
    RC_PROTO_CEC       = 22,
    RC_PROTO_IMON      = 23,
    RC_PROTO_RCMM12    = 24,
    RC_PROTO_RCMM24    = 25,
    RC_PROTO_RCMM32    = 26,
    RC_PROTO_XBOX_DVD  = 27,
};

#endif
```

ir-tx.c文件代码如下：

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <poll.h>
#include <signal.h>
#include <sys/types.h>
#include <linux/types.h>
#include <pthread.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include "ir.h"

#define NEC_NBITS      32
#define NEC_UNIT       562500  /* ns */
#define NEC_HEADER_PULSE    (16 * NEC_UNIT) /* 9ms */
#define NECX_HEADER_PULSE    (8  * NEC_UNIT) /* Less common NEC variant */
#define NEC_HEADER_SPACE    (8  * NEC_UNIT) /* 4.5ms */
#define NEC_REPEAT_SPACE    (4  * NEC_UNIT) /* 2.25ms */
#define NEC_BIT_PULSE       (1  * NEC_UNIT)
#define NEC_BIT_0_SPACE     (1  * NEC_UNIT)
```

```
#define NEC_BIT_1_SPACE      (3  * NEC_UNIT)
#define   NEC_TRAILER_PULSE   (1  * NEC_UNIT)
#define   NEC_TRAILER_SPACE   (10 * NEC_UNIT) /* even longer in reality */

#define NS_TO_US(nsec)      ((nsec) / 1000)

#define GPIO_IR_RAW_BUF_SIZE   128
#define DEFAULT_CARRIER_FREQ   38000
#define DEFAULT_DUTY_CYCLE   33
#define NEC_ADDR_SHIFT      24
#define NEC_UNADDR_SHIFT   16
#define NEC_CMD_SHIFT      8
#define NEC_UNCMD_SHIFT      0
#define NEC_ENCODE_MASK      0xff
#define RAW_MASK      0xffffff
#define VERSION_NUM      256
#define RAW_BANK      8
#define SHIFT_MASK      8

uint32_t rx_raw_buf[GPIO_IR_RAW_BUF_SIZE];
uint32_t tx_raw_buf[GPIO_IR_RAW_BUF_SIZE];

static int fd;
static int fd1;
struct pollfd poll_fds[1];
static int int_exit;
static int kernel_flag;
pthread_t tid;

static void print_usage(const char *argv0)
{   printf("usage: %s [options]\n", argv0);
   printf("\n");
   printf("gpio ir receive test:\n");
   printf("\tgpio_ir_test rx\n");
   printf("\n");
   printf("gpio ir send test:\n");
   printf("\tgpio_ir_test tx <code>\n");
   printf("\n");
   printf("gpio ir loop test, rx&tx:\n");
   printf("\tgpio_ir_test loop\n");
   printf("\n");
}

static int nec_modulation_byte(uint32_t *buf, uint8_t code)
{
   int i = 0;
   uint8_t mask = 0x01;

   while (mask) {
      /*low bit first*/
      if (code & mask) {
         /*bit 1*/
         *(buf + i) = LIRC_PULSE(NS_TO_US(NEC_BIT_PULSE));
         *(buf + i + 1) = LIRC_SPACE(NS_TO_US(NEC_BIT_1_SPACE));
      } else {
         /*bit 0*/
         *(buf + i) = LIRC_PULSE(NS_TO_US(NEC_BIT_PULSE));
         *(buf + i + 1) = LIRC_SPACE(NS_TO_US(NEC_BIT_0_SPACE));
      }
      mask <<= 1;
```

```
        i += 2;
    }
    return i;
}

static int nec_ir_encode(uint32_t *raw_buf, uint32_t key_code)
{
    uint8_t address, not_address, command, not_command;
    uint32_t *head_p, *data_p, *stop_p;

    address    = (key_code >> NEC_ADDR_SHIFT) & NEC_ENCODE_MASK;
    not_address = (key_code >> NEC_UNADDR_SHIFT) & NEC_ENCODE_MASK;
    command    = (key_code >>  NEC_CMD_SHIFT) & NEC_ENCODE_MASK;
    not_command = (key_code >>  NEC_UNCMD_SHIFT) & NEC_ENCODE_MASK;

    /*head bit*/
    head_p = raw_buf;
    *(head_p) = LIRC_PULSE(NS_TO_US(NEC_HEADER_PULSE));
    *(head_p + 1) = LIRC_SPACE(NS_TO_US(NEC_HEADER_SPACE));

    /*data bit*/
    data_p = raw_buf + 2;
    nec_modulation_byte(data_p,  address);

    data_p += 16;
    nec_modulation_byte(data_p,  not_address);

    data_p += 16;
    nec_modulation_byte(data_p,  command);

    data_p += 16;
    nec_modulation_byte(data_p,  not_command);

    /*stop bit*/
    stop_p = data_p + 16;
    *(stop_p) = LIRC_PULSE(NS_TO_US(NEC_TRAILER_PULSE));
    *(stop_p + 1) = LIRC_SPACE(NS_TO_US(NEC_TRAILER_SPACE));

    /*return the total size of nec protocal pulse*/
    /* linux-5.4 needs 67 byte datas */
    if (kernel_flag == 1)
        return ((NEC_NBITS + 2) * 2 - 1);
    else
        return ((NEC_NBITS + 2) * 2);
}

void *ir_recv_thread(void *arg)
{
    int size = 0, size_t = 0;
    int i = 0;
    int dura;
    int ret;
    int total = 0;

    poll_fds[0].fd = fd;
    poll_fds[0].events = POLLIN | POLLERR;
    poll_fds[0].revents = 0;
    while (!int_exit) {
        ret = poll(poll_fds, 1, 12);
        if (!ret) {
```

```
            printf("\n--------------------\n");
            total = 0;
        } else {
            if (poll_fds[0].revents == POLLIN) {
                size = read(fd, (char *)(rx_raw_buf),
                        GPIO_IR_RAW_BUF_SIZE);
                size_t = size / sizeof(uint32_t);
                for (i = 0; i < size_t; i++) {
                    dura = rx_raw_buf[i] & RAW_MASK;
                    printf("%d ", dura);
                    if ((total++) % RAW_BANK == 0)
                        printf("\n");
                }
            }
        }
    }

    return NULL;
}

void gpio_ir_test_close(int sig)
{
    /* allow the stream to be closed gracefully */
    signal(sig, SIG_IGN);
    int_exit = 1;
    if (fd1 != fd)
        close(fd1);
    close(fd);
}

int main(int argc, char **argv)
{
    int ret;
    int size = 0, size_t = 0;
    int i = 0;
    int duty_cycle;
    int carrier_freq;
    int key_code = 0;
    int err = 0;
    int cnt = 0;
    /* use for check the current kernel version */
    int fd_version;
    char version_buf[VERSION_NUM];

    fd_version = open("/proc/version", O_RDONLY);
    if (fd_version < 0) {
        printf("The system is not mount proc, please check\n");
        return -1;
    }
    ret = read(fd_version, version_buf, sizeof(version_buf));
    if (ret < 0) {
        printf("Can't not read /proc/verison\n");
        return -1;
    }
    switch(version_buf[14]) {
        case '4':
            kernel_flag = 0;
            printf("This is 4.* linux kernel\n");
            break;
        case '5':
```

```
        kernel_flag = 1;
        printf("This is 5.* linux kernel\n");
        break;
    default:
        kernel_flag = 0;
        printf("This is %c.* kernel which is not support\n", version_buf[14]);
        break;
}
close(fd_version);

/* catch ctrl-c to shutdown cleanly */
signal(SIGINT, gpio_ir_test_close);

if (argc < 2) {
    print_usage(argv[0]);
    return -1;
}

fd = open("/dev/lirc0", O_RDWR);
if (fd < 0) {
    printf("can't open lirc0 recv, check driver!\n");
    return 0;
} else {
    printf("lirc0 open succeed.\n");
}

fd1 = open("/dev/lirc1", O_RDWR);
if (fd1 < 0) {
    printf("can't open lirc1 send, check driver!\n");
    fd1 = fd;
    printf("There is no lirc1, use the lirc0 as lirc1.\n");
} else {
    printf("lirc1 open succeed.\n");
}

for (i = 1; i < argc; i++) {
    if (!strcmp(argv[i], "-n")) {
        if (i + 1 >= argc) {
            printf("Option -n expects an argument.\n\n");
            print_usage(argv[0]);
            goto OUT;
        }
        continue;
    }
}

if (!strcmp(argv[1], "rx")) {
    err = pthread_create(&tid, NULL, (void *)ir_recv_thread, NULL);
    if (err != 0) {
        printf("create pthread error: %d\n", __LINE__);
        goto OUT;
    }

    do {
        usleep(1000);
    } while (!int_exit);

} else if (!strcmp(argv[1], "tx")) {
    if (argc < 3) {
        fprintf(stderr, "No data passed\n");
```

```
    goto OUT;
}

if (sscanf(argv[2], "%x", &key_code) != 1) {
    fprintf(stderr, "no input data: %s\n", argv[2]);
    goto OUT;
}

key_code = key_code << SHIFT_MASK;

duty_cycle = DEFAULT_DUTY_CYCLE;
if (ioctl(fd1, LIRC_SET_SEND_DUTY_CYCLE, &duty_cycle)) {
    fprintf(stderr,
        "lirc0: could not set carrier duty: %s\n",
        strerror(errno));
    goto OUT;
}

if (sscanf(argv[3], "%ul", &carrier_freq) != 1) {
    fprintf(stderr, "no input data: %s\n", argv[3]);
    goto OUT;
}

if (ioctl(fd1, LIRC_SET_SEND_CARRIER, &carrier_freq)) {
    fprintf(stderr,
        "lirc0: could not set carrier freq: %s\n",
        strerror(errno));
    goto OUT;
}

printf("irtest: send key code : 0x%x\n", key_code);

size = nec_ir_encode(tx_raw_buf, key_code);
/*dump the raw data*/
for (i = 0; i < size; i++) {
    printf("%d ", *(tx_raw_buf + i) & RAW_MASK);
    if ((i + 1) % RAW_BANK == 0)
        printf("\n");
}
printf("\n");

/* linux-5.4 IR core should delete bit24~bit31 */
if (kernel_flag == 1) {
    for (i = 0; i < size; i++) {
        tx_raw_buf[i] = (tx_raw_buf[i] & RAW_MASK);
    }
}

if (argc > 4 && !strcmp(argv[3], "-n")) {
            cnt = atoi(argv[4]);
            for (i = 0; i < cnt; i++) {
                size_t = size * sizeof(uint32_t);
                ret = write(fd1, (char *)tx_raw_buf, size_t);
                if (ret > 0)
                    printf("irtest No.%d: send %d bytes ir raw data\n\n",
                        i, ret);
        else
            printf("irtest No.%d: send %d failed!\n", i, ret);
        usleep(100*1000);
            }
```

```
        } else {
            size_t = size * sizeof(uint32_t);
            ret = write(fd1, (char *)tx_raw_buf, size_t);
            if (ret > 0)
                printf("irtest: send %d bytes ir raw data\n\n", ret);
            else
                printf("irtest: send %d failed\n", ret);
        }

    } else if (!strcmp(argv[1], "loop")) {

        /*code: 0x13*/
        key_code = 0x04fb13ec;
        err = pthread_create(&tid, NULL, (void *)ir_recv_thread, NULL);
        if (err != 0) {
            printf("create pthread error: %d\n", __LINE__);
            goto OUT;
        }

        duty_cycle = DEFAULT_DUTY_CYCLE;
        if (ioctl(fd1, LIRC_SET_SEND_DUTY_CYCLE, &duty_cycle)) {
            fprintf(stderr,
                "lirc0: could not set carrier duty: %s\n",
                strerror(errno));
            goto OUT;
        }
        carrier_freq = DEFAULT_CARRIER_FREQ;
        if (ioctl(fd1, LIRC_SET_SEND_CARRIER, &carrier_freq)) {
            fprintf(stderr,
                "lirc0: could not set carrier freq: %s\n",
                strerror(errno));
            goto OUT;
        }

        /*echo 50ms transmit one frame */

        if (argc > 3 && !strcmp(argv[2], "-n")) {
            cnt = atoi(argv[3]);
            for (i = 0; i < cnt; i++) {
                size = nec_ir_encode(tx_raw_buf, key_code);
                size_t = size * sizeof(uint32_t);
                /* linux-5.4 IR core should delete bit24~bit31 */
                if (kernel_flag == 1) {
                    for (i = 0; i < size; i++) {
                        tx_raw_buf[i] = (tx_raw_buf[i] & RAW_MASK);
                    }
                }
                printf("send key code : 0x%x, No.%d\n", key_code, i);
                write(fd1, (char *)tx_raw_buf, size_t);
                usleep(100*1000);
                if (int_exit)
                    break;
            }
        } else {
                i = 0;
            do {
                size = nec_ir_encode(tx_raw_buf, key_code);
                size_t = size * sizeof(uint32_t);
                /* linux-5.4 IR core should delete bit24~bit31 */
                if (kernel_flag == 1) {
```

```
                for (i = 0; i < size; i++) {
                    tx_raw_buf[i] = (tx_raw_buf[i] & RAW_MASK);
                }
            }
            printf("send key code : 0x%x, %d\n", key_code, i++);
            write(fd1, (char *)tx_raw_buf, size_t);
            usleep(100*1000);
        } while (!int_exit);
    }
  } else if (!strcmp(argv[1], "-n")) {
    print_usage(argv[0]);
  }

OUT:
  if (fd1 != fd)
    close(fd1);
  close(fd);
  return 0;
}
```

该 Demo 可以用来调用 TX 模块发送数据信号. 测试命令：tx 0x04 3800 注：Demo 采用 NEC 协议进行编码，设置载波占空比为 33%，设置载波频率为 38KHz。

著作权声明

商标声明

免责声明