NanoPI2-LCD 驱动移植

2018.4.3

周星宇、闫隆鑫（ylx601@gmail.com）

目录

1.LCD 触摸屏浅析

一个 Linux 的触摸屏驱动其实包含三个部分：

1. LCD 设备驱动，负责图像显示的部分；
2. 触摸驱动，负责触摸屏下触摸板的响应；
3. 背光驱动，负责显示屏亮度的调节。

本文的余下部分将会分别从这三个部分介绍电阻触摸屏在 Android 系统上的移植细节。

- 电阻触摸屏型号：AM-800600P4TMQW-TB0H
- Android 源码版本：Android 5.1
- Android 开发板型号：NanoPI T2(友善之臂提供)

2.LCD 设备

2.1 LCD 设备描述

NanoPi2 支持的所有 LCD 型号使用一个结构体来存储 nanopi2_lcd_config

arch\arm\plat-s5p4418\nanopi2\lcds.c

```
/* Try to guess LCD panel by kernel command line, or
 * using *HD101* as default
 * \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\lcds.c
*/
static struct {
    char *name;
    struct nxp_lcd *lcd;
    int ctp;
} nanopi2_lcd_config[] = {
    { "HD101",  &wxga_hd101, 1 },
```

```
    { "HD101B", &wxga_hd101, CTP_GOODIX  },
    { "HD700",  &wxga_hd700, 1 },
    { "HD702",  &wxga_hd700, CTP_GOODIX  },
    { "S70",    &wvga_s70,   1 },
    { "S702",   &wvga_s702,  1 },
    { "S70D",   &wvga_s70d,  0 },
    { "X710",   &wsvga_x710, CTP_ITE7260 },
    { "S430",   &wvga_s430,  CTP_HIMAX   },
#ifndef CONFIG_ANDROID
    { "H43",    &hvga_h43,   0 },
    { "P43",    &hvga_p43,   0 },
    { "W35",    &qvga_w35,   0 },
#endif
    /* TODO: Testing */
    { "W50",    &wvga_w50,   0 },
    { "W101",   &wsvga_w101, 1 },
    { "A97",    &xga_a97,    0 },
    { "LQ150",  &xga_lq150,  1 },
    { "L80",    &vga_l80,    1 },
    { "BP101",  &wxga_bp101, 1 },
    { "HDMI",   &hdmi_def,   0 },   /* Pls keep it at last */
};
```

其中存储了每种 LCD 的名称与驱动参数。

第一列：LCD 名字

第二列：LCD 参数结构体

第三列：使用触摸芯片型号

下面用 wsvga_x710 型号的 LCD 举例。

```
/*
 * \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\lcds.c
 */
static struct nxp_lcd wsvga_x710 = {
    .width= 1024,
    .height = 600,
    .p_width = 154,
    .p_height = 90,
    .bpp = 24,
    .freq = 61,
```

```
    .timing = {
        .h_fp = 84,
        .h_bp = 84,
        .h_sw = 88,
        .v_fp = 10,
        .v_fpe = 1,
        .v_bp = 10,
        .v_bpe = 1,
        .v_sw = 20,
    },
    .polarity = {
        .rise_vclk = 0,
        .inv_hsync = 1,
        .inv_vsync = 1,
        .inv_vden = 0,
    },
    .gpio_init = hd101_gpio_init,
};
```

其中各参数的含义如下：

```
/*
 * \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\include
 * struct nxp_lcd
 * @width:      horizontal resolution <水平分辨率>
 * @height:     vertical resolution <垂直分辨率>
 * @p_width:    width of lcd in mm <水平物理尺寸>
 * @p_height:   height of lcd in mm <垂直物理尺寸>
 * @bpp:        bits per pixel <每个像素要用多少位(24)>
 * @freq:       vframe frequency <刷新帧率(61)>
 * @timing:     timing values <存储时序参数的结构体>
 * @polarity:   polarity settings <结构体，存储信号线极性>
 * @gpio_init:  pointer to GPIO init function <GPIO 初始化函数>
 *
 */
struct nxp_lcd {
    int width;
    int height;
    int p_width;
    int p_height;
    int bpp;
    int freq;
    struct  nxp_lcd_timing timing;
```

```
    struct  nxp_lcd_polarity polarity;
    void    (*gpio_init)(void);
};
```

LCD 时序参数结构体如下：

```
/*
 * struct nxp_lcd_timing
 * @h_fp:   horizontal front porch
 * @h_bp:   horizontal back porch
 * @h_sw:   horizontal sync width
 * @v_fp:   vertical front porch
 * @v_fpe:  vertical front porch for even field
 * @v_bp:   vertical back porch
 * @v_bpe:  vertical back porch for even field
 */
struct nxp_lcd_timing {
    int h_fp;
    int h_bp;
    int h_sw;
    int v_fp;
    int v_fpe;
    int v_bp;
    int v_bpe;
    int v_sw;
};
```

时序参数的含义参考 LCD 控制时序。

参数确定方法见《LCD 驱动时序参数的确定.docx》

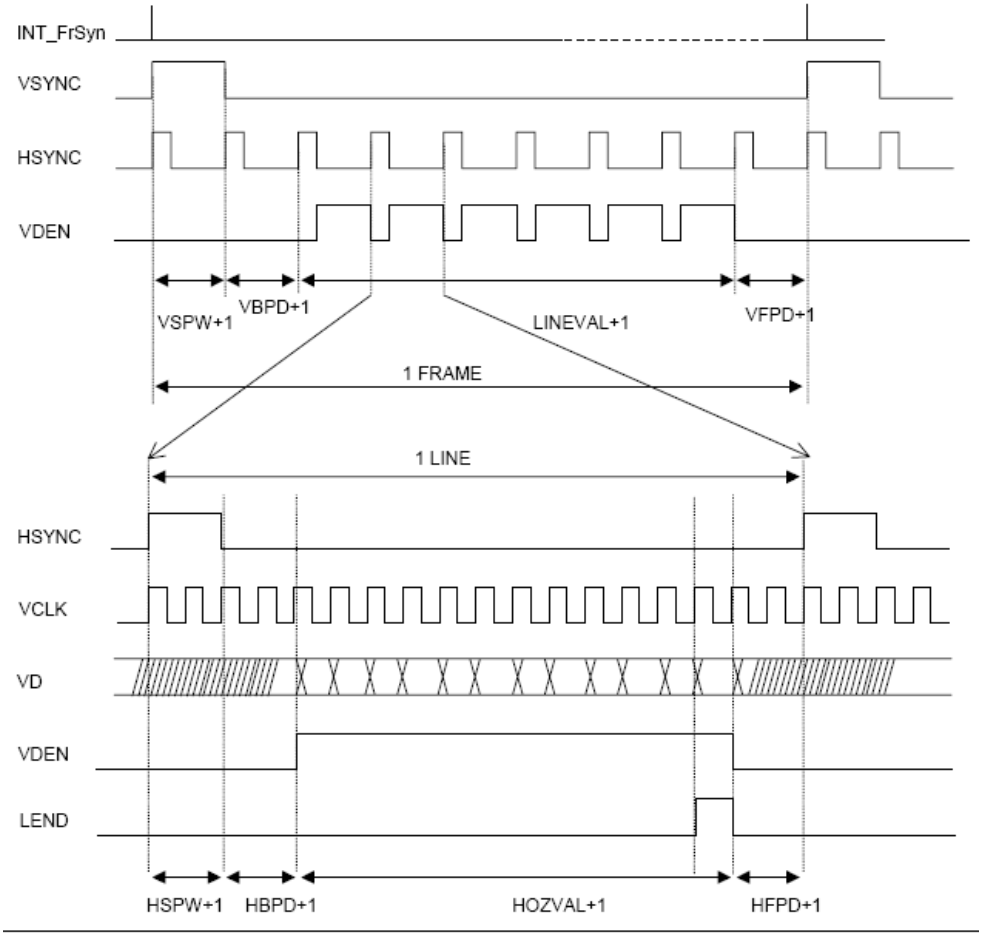

Figure 22-7. LCD RGB interface Timing

信号线极性结构体定义了个别信号线是否反转。

是否反转也可以在 LCD 时序图中看出来

```
/*
 * struct nxp_lcd_polarity
 * @rise_vclk:  if 1, video data is fetched at rising edge
 * @inv_hsync:  if HSYNC polarity is inversed
 * @inv_vsync:  if VSYNC polarity is inversed
 * @inv_vden:   if VDEN polarity is inversed
 */
struct nxp_lcd_polarity {
    int rise_vclk;
    int inv_hsync;
    int inv_vsync;
    int inv_vden;
};
```

GPIO 初始化函数中定义了引脚的输出强度

2.2 驱动移植

1) 根据数据手册《附件 3 AM-800600P4TMQW-TB0H(TTL 接口 4 线电阻触摸 400 流明 带背光驱动 》》中的相关数据定义我们所用型号的 LCD 描述结构体：

```
static struct nxp_lcd wsvga_l601 = {
  .width= 800,
  .height = 600,
  .p_width = 163,
  .p_height = 122,
  .bpp = 24,
  .freq = 60,

  .timing = {
   .h_fp = 112,
   .h_bp = 40,
   .h_sw = 48,
   .v_fp = 21,
   .v_fpe = 1,
   .v_bp = 36,
   .v_bpe = 1,
   .v_sw = 3,
  },
  .polarity = {
   .rise_vclk = 1,
   .inv_hsync = 1,
   .inv_vsync = 1,
   .inv_vden = 0,
  },
  .gpio_init = s70_gpio_init,
  //.gpio_init = hd101_gpio_init,
};

```

2) 把我们刚刚定义的结构体添加进 nanopi2_lcd_config 中，我们把它添加到了下标为 8 的位

置；

```
static struct {
 char *name;
 struct nxp_lcd *lcd;
 int ctp;
} nanopi2_lcd_config[] = {
 { "HD101", &wxga_hd101, 1 },
 { "HD101B", &wxga_hd101, CTP_GOODIX  },
 { "HD700", &wxga_hd700, 1 },
 { "HD702", &wxga_hd700, CTP_GOODIX  },
 { "S70", &wvga_s70,   1 },
 { "S702", &wvga_s702,  1 },
 { "S70D", &wvga_s70d,  0 },
 { "X710", &wsvga_x710, CTP_ITE7260 },
 {"L601", &wsvga_l601, 0},
 { "S430", &wvga_s430,  CTP_HIMAX   },
```

3) 修改 nanopi2_setup_lcd 函数，这个函数的原理是根据 1-wire 协议读取到的 lcd 屏的名称，将名称与 nanopi2_lcd_config 中的字符串对比，选择名称一样的结构体进行加载。但是我们使用的屏幕不支持 1-wire 协议，所以我们在函数中将加载的 lcd 描述结构体标号强制赋值为 8；

```
static int __init nanopi2_setup_lcd(char *str)
{
 char *delim;
 int i;

 delim = strchr(str, ',');
 if (delim)
  *delim++ = '\0';

 if (!strncasecmp("HDMI", str, 4)) {
  goto __ret;//ylx add
  struct hdmi_config *cfg = &nanopi2_hdmi_config[0];
  struct nxp_lcd *lcd;

  lcd_idx = ARRAY_SIZE(nanopi2_lcd_config) - 1;
  lcd = nanopi2_lcd_config[lcd_idx].lcd;
```

```
17.
18.    for (i = 0; i < ARRAY_SIZE(nanopi2_hdmi_config); i++, cfg++) {
19.     if (!strcasecmp(cfg->name, str)) {
20.      lcd->width = cfg->width;
21.      lcd->height = cfg->height;
22.      goto __ret;
23.     }
24.    }
25.   }
26.
27.   for (i = 0; i < ARRAY_SIZE(nanopi2_lcd_config); i++) {
28.    goto __ret;//ylx add
29.    if (!strcasecmp(nanopi2_lcd_config[i].name, str)) {
30.     lcd_idx = i;
31.     break;
32.    }
33.   }
```

3.触摸

3.1 编译驱动

修改.config 文件，添加触摸屏驱动支持；更方便的方法是在 make menuconfig 中修改 Linux 内核编译要加载的驱动模块；具体路径为：Device Drivers→Input device support→Touchscreens

注意：为了避免触摸屏使用的 I2C 地址发生冲突，安全起见除了要保留的触摸屏驱动 tsc2007 其他驱动全部取消。

```
1.  # CONFIG_TOUCHSCREEN_TOUCHIT213 is not set
2.  # CONFIG_TOUCHSCREEN_TSC_SERIO is not set
3.  # CONFIG_TOUCHSCREEN_TSC2005 is not set
4.  CONFIG_TOUCHSCREEN_TSC2007=y
5.  # CONFIG_TOUCHSCREEN_ST1232 is not set
6.  # CONFIG_TOUCHSCREEN_TPS6507X is not set
```

3.2 定义 I2c 设备

1) 定义 I2C 设备驱动的相关结构体

触摸屏相关的 I2C 设备在

linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\device.c

中定义。

```
1.  //ylx add
2.  #if defined(CONFIG_TOUCHSCREEN_TSC2007)
3.  #define  TSC2007_I2C_BUS    (2)
4.  static int tsc2007_get_pendown_state(void){
5.    return !gpio_get_value(CFG_IO_TOUCH_IRQ);
6.  }
7.
8.  struct tsc2007_platform_data tsc2007_data ={
9.    .model = 2007,
10.   .x_plate_ohms = 180,
11.   .get_pendown_state = tsc2007_get_pendown_state,
12. };
13.
14. static struct i2c_board_info __initdata tsc2007_i2c_bdi = {
15.   I2C_BOARD_INFO("tsc2007", (0x90>>1)),
16.   .irq = PB_PIO_IRQ(CFG_IO_TOUCH_IRQ),
17.   .platform_data = &tsc2007_data,
18. };
19. #endif
```

其中需要注意：

a) I2C 设备的地址需要改成触摸芯片手册中的(例子中 tsc2007 为 0x90>>1)

b) "tsc2007"为 I2C 设备名称，需要和驱动 tsc2007 设备驱动程序中的驱动名称一致,具体的驱动设备名称可以在\drivers\input\touchscreen\tsc2007.c 中的 tsc2007_probe 函数中查看。`input_dev->name = "tsc2007";`

c) 不同触摸芯片型号需要填充 I2c_board_info 结构体中的不同成员，(如 tsc2007 需要填

充 device 成员)。

d) tsc2007_platform_data 的定义包含在在头文件中，记得在程序头部加入

```
#include <linux/i2c/tsc2007.h>
```

2) 注册 I2C 设备

在 nxp_board_devices_register 函数中添加

```
1.  //ylx add
2.  #if defined(CONFIG_TOUCHSCREEN_TSC2007)
3.    printk("plat: add touch(tsc2007) device\n");
4.    i2c_register_board_info(TSC2007_I2C_BUS, &tsc2007_i2c_bdi, 1);
5.  #endif
6.
```

3.3 修改驱动程序

驱动程序所在路径为： \drivers\input\touchscreen\tsc2007.c

设备驱动程序一般情况下 linux 内核的 drivers 目录中会自带，也很少需要我们去修改。只要修改.config 文件将它们编译进内核就可以了。但是在本例中还是要针对实际情况略作修改：

1) 修改屏幕分辨率，根据实际使用的 LCD 分辨率修改相应的宏定义：(本例中的 LCD 分辨率为 800x600)

```
1.  #define LCD_SCREEN_X_PIXEL  800 //1024
2.  #define LCD_SCREEN_Y_PIXEL  600 //600
```

2) 修改 CFG_IO_TOUCH_PENDOWN_DETECT，这个宏定义对应的是电阻触摸芯片 pendown 脚所连接的 CPU 的引脚，作用是感应触摸操作并发起相应的中断请求；我们可以通过查阅板子的原理图，找到这个 pendown 实际对应的 CPU 的引脚。在本例中实际 对应的引脚与宏定义 CFG_IO_TOUCH_IRQ 所定义的引脚一致。

```
1.  #define CFG_IO_TOUCH_PENDOWN_DETECT CFG_IO_TOUCH_IRQ
```

3) 修改软中断处理函数 tsc2007_soft_irq。先把驱动烧写如板子里，试验一下。看实际操作中，x,y 坐标的移动方向是否与实际操作相反。根据实际情况决定 x\y 是否要反向。

```
tc.x = LCD_SCREEN_X_PIXEL*(x)/(ADCVAL_X_MAX -ADCVAL_X_MIN);
 tc.y = LCD_SCREEN_Y_PIXEL*(y)/(ADCVAL_Y_MAX-ADCVAL_Y_MIN);
tc.x = LCD_SCREEN_X_PIXEL - tc.x; //x 反向
tc.y = LCD_SCREEN_Y_PIXEL - tc.y; //y 反向
```

4) 校准 x、y 坐标的最大最小值

```
#define ADCVAL_X_MIN    120 //150
#define ADCVAL_X_MAX    3970 //3950
#define ADCVAL_Y_MIN    190 //280
#define ADCVAL_Y_MAX    3930 //3850
```

校准方法：

a) 在 tsc2007_soft_irq 中在对 tc.x、tc.y 做处理前，添加以下代码：

```
//ylx add
      printk(
        "DOWN point(%4d,%4d), pressure (%4u)\n",
        tc.x, tc.y, rt);
```

b) 使用 adb shell 进入 Android 系统，输入 dmesg 查看内核信息。

c) 分别点击屏幕的上下左右边缘，使用 dmesg 查看 log 中打印的坐标，分别选出 x、y 的最大最小值，把它写入 tsc2007.c 中的宏定义里。

下面再介绍一个移植设备驱动的诀窍

诀窍：如何编写设备注册程序可参考 linux 内核源码中未编译进内核的其他平台的实现(如 linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\mach-imx\mach-cpuimx51sd.c 中就有 tsc2007 设备的注册，如下图所示)

```
//linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\mach-imx\mach-cpuimx51sd.c
static struct tsc2007_platform_data tsc2007_info = {
```

```
3.      .model          = 2007,
4.      .x_plate_ohms       = 180,
5.      .get_pendown_state  = ts_get_pendown_state,
6.      .init_platform_hw   = ts_init,
7.  };
8.
9.  static struct i2c_board_info tsc_device = {
10.     I2C_BOARD_INFO("tsc2007", 0x48),
11.     .type       = "tsc2007",
12.     .platform_data  = &tsc2007_info,
13.     /*.irq is selected on ap4evb_init */
14. };
```

这样一来器件 I2C 地址也不需要去芯片手册找了。

3.3 修改 IDC 文件

如果是 linux 系统，上面的工作做完之后已经 ok。

但是 android 系统上还需要修改.idc 文件，如果不修改的话会变成触摸鼠标

编写 tsc2007.idc 如下，将其 adb push 入 Android 系统的 system/usr/idc 目录下，也可放在 out/target/product/nanopi2/system/usr/idc 下重新编译镜像，重新烧写板子。

```
1.  touch.deviceType = touchScreen
2.  touch.orientationAware = 1
3.
4.  # Size
5.  touch.size.calibration = diameter
6.  touch.size.scale = 10
7.  touch.size.bias = 0
8.  touch.size.isSummed = 0
9.
10. # Pressure
11. touch.pressure.calibration = amplitude
12. touch.pressure.scale = 0.005
13.
14. # Orientation
15. touch.orientation.calibration = none
```

参考：

http://blog.chinaunix.net/uid-149881-id-4347448.html

4.背光驱动

Linux 提供了专门的 backlight 驱动框架，根据实际情况更改里面的 PWM 接口即可。

由于我们所使用的 LCD 屏本身亮度较低，在实际使用中要一直将亮度置为最高，所以直接将背光引脚接至 5V 直流上（相当于 100%PWM）

这部分驱动的移植，读者可以自行查阅相关资料开展。

5.参考资料

- 关于怎么编译 Linux 内核 和 Android 镜像可以参考：
  - http://wiki.friendlyarm.com/wiki/index.php/NanoPC-T2/zh
  - ref/ Android 镜像编译补充说明.pdf
- 随本文档的文件夹 ref/中的所有文档
- https://github.com/ylxseu/linux3.4y-lab601 最终版的 Linux 内核源码