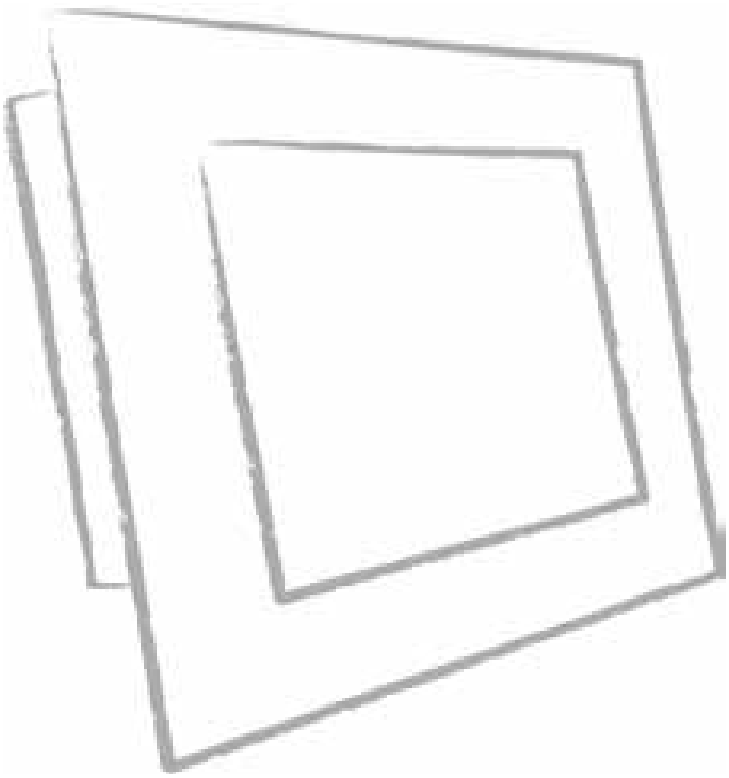
User's Manua

4418 6818 drivers



Winsonic®

WINSONIC ELECTRONICS Co., LTD
No.290-1, Wen Chung Rd., Taoyuan City, Taiwan, R.O.C
Tel:+886-3-3704789
Fax:+886-3-3704722
E-mail:sales@ewinsonic.com
www.ewinsonic.com

Compile Path Code

LCD：lcd compile path: kernel\drivers\video mainly include-- nxp-fb.c ；resolution compile path: cfg_main.h（uboot&kernel）。

6818：

uboot u-boot\board\s5p6818\include\cfg_main.h

kernel kernel\arch\arm\plat-s5p6818\drone\include\ cfg_main.h

4418：

uboot u-boot\board\s5p4418\include\cfg_main.h

kernel kernel\arch\arm\plat-s5p4418\drone\include\ cfg_main.h

VGA：Modify LCD

TP：Touch compile path kernel\drivers\input\touchscreen ft5x0x_ts.c gt9xx.c

Camera：compile path kernel\drivers\media\video ov5645.c

Audio：compile path kernel\sound\soc\codecs\wm8960.c and kernel\sound\soc\nexell\nxp-wm8960.c

RTL8211E：ETHERNET compile path kernel\drivers\net\ethernet\nexell\nxpmac

nxpmac_main.c Button：compile path kernel\drivers\input\keyboard Nxp_io_key.c

SDMMC：sdmmc compile path kernel\drivers\mmc

UART：compile path kernel\drivers\tty\serial

RTC：RTC compile path kernel\drivers\rtc rtc-dev.c

HDMI：HDMI compile path kernel\drivers\media\video\nexell\out

USB：usb compile path kernel\drivers\usb (inculde OTG)

Gerenal Documents (kernel directory)：

6818：

Device.c (arch\arm\plat-s5p6818\drone) Cfg_main.h (arch\arm\plat-s5p6818\drone\include) Devices.c (arch\arm\mach-s5p6818)

4418：

Device.c (arch\arm\plat-s5p4418\drone) Cfg_main.h (arch\arm\plat-s5p4418\drone\include) Devices.c (arch\arm\mach-s5p4418)

Rp_gpio_ctrl.c (drivers\rongpin) Ggpio control code.

Ft5x0x_ts.c Gt9xx.c (drivers\input\touchscreen) Touch Panel driver.

I2c-gpio.c (drivers\i2c\busses) I2C bus-driving.

Nxp_io_key.c (drivers\input\keyboard) Button driver.

Nxp-capture.c (drivers\media\video\nexell\capture) Camera driver.

Nxp-vin-clipper.c (drivers\media\video\nexell\capture) Camera.

spi-slsi.c(drivers\spi\) SPI bus-driving

GPIO Function Deployment

- PCB GPIO PIN define
- GPIO Output
- GPIO Input
- GPIO Reading
- GPIO Interrupt

PCB GPIO PIN Define：

4418 6818 Kernel, GPIO pin define has ABCDE categories, each category has 32 . PIN Define as below：

```
/* gpio group pad start num. */
enum {
    PAD_GPIO_A      = (0 * 32),
    PAD_GPIO_B      = (1 * 32),
    PAD_GPIO_C      = (2 * 32),
    PAD_GPIO_D      = (3 * 32),
    PAD_GPIO_E      = (4 * 32),
    PAD_GPIO_ALV    = (5 * 32),
};
```

Example：

GPIOB29 to PAD_GPIO_B + 29

Hardware naming：GPIOB29

Softwawre naming：PAD_GPIO_B + 29

GPIOC10 to PAD_GPIO_C + 10

Hardware naming：GPIOC10

Softwawre naming：PAD_GPIO_C + 10

GPIO output

GPIO output Value Function: int gpio_direction_output(unsigned gpio, int value)

```
int gpio_direction_output(unsigned gpio, int value)
```

unsigned gpio：GPIO output value；int value：(0 Low Level，1 High Level）
High level example: gpio_direction_output(PAD_GPIO_C + 10, 1);
Low level example：gpio_direction_output(PAD_GPIO_C + 10, 0);

GPIO Input

GPIO Input Function :
int gpio_direction_input(unsigned gpio)

```
int gpio_direction_input(unsigned gpio)
```

GPIOC10 to Input function :
gpio_direction_input（PAD_GPIO_C + 10）；

GPIO Reading

Reaing :
static inline int gpio_get_value(unsigned gpio)

Example :
GPIOC10 value
gpio_get_value(PAD_GPIO_C + 10)

GPIO Interrupt

```
void nxp_soc_gpio_set_int_enable(unsigned int io, int on)
```

Parameter：

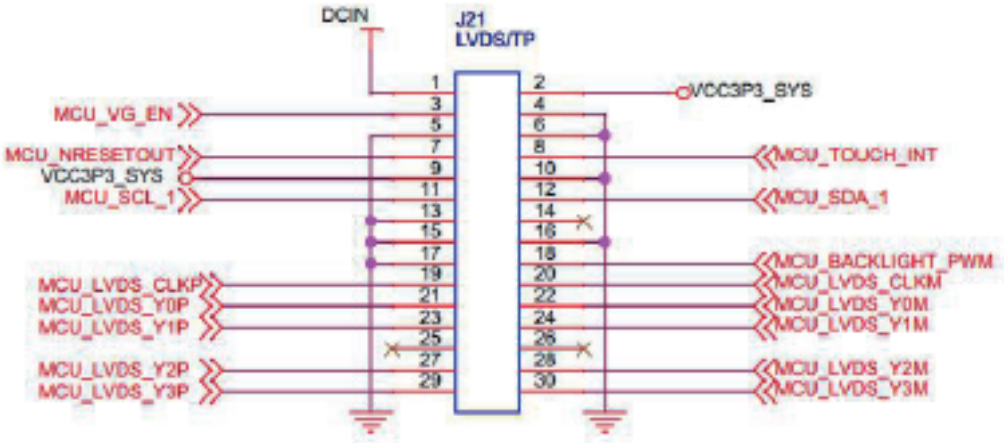
unsigned io ：single GPIO pin

int on：value 1: interrupt，value 0: can not interrupt

Touch panel is interrupt function :

Gt9xx.c (drivers\input\touchscreen)

Use MCU_TOUCH_INT interrupt pin



MCU_TOUCH_INT to GPIO Pin is GPIOB29

How to set GPIOB29 to interrupt

```
#define CFG_IO_TOUCH_PENDOWN_DETECT          (PAD_GPIO_B + 29)

#define GTP_INT_PORT     CFG_IO_TOUCH_PENDOWN_DETECT
```

GTP_GPIO_REQUEST(GTP_INT_PORT, “GTP_INT_IRQ”);

gpio_direction_input(GTP_INT_PORT);

nxp_soc_gpio_set_int_enable(GTP_INT_PORT,1);

GTP_GPIO_FREE(GTP_INT_PORT);

References :

irq.h (include\linux)

```
enum {
    IRQ_TYPE_NONE        = 0x00000000,
    IRQ_TYPE_EDGE_RISING    = 0x00000001,
    IRQ_TYPE_EDGE_FALLING   = 0x00000002,
    IRQ_TYPE_EDGE_BOTH   = (IRQ_TYPE_EDGE_FALLING | IRQ_TYPE_EDGE_RISING),
    IRQ_TYPE_LEVEL_HIGH = 0x00000004,
    IRQ_TYPE_LEVEL_LOW   = 0x00000008,
    IRQ_TYPE_LEVEL_MASK = (IRQ_TYPE_LEVEL_LOW | IRQ_TYPE_LEVEL_HIGH),
    IRQ_TYPE_SENSE_MASK = 0x0000000f,
    IRQ_TYPE_DEFAULT    = IRQ_TYPE_SENSE_MASK,
```

How to set up parameter to touch function

request_irq(ts->client->irq,goodix_ts_irq_handler,
IRQ_TYPE_EDGE_FALLING, "goodix_ts", goodix_ts);

Compiles

1. LCD, TP

1. S5P6818 support LCD, TP（Touch Panel）

LCD source code ：

LVDS 7 inch 1024*600

LVDS 10 inch 1024*600

LVDS 10 inch 1280*800

MIPI 10 inch 1920*1200

2. LCD Resolution：

A. Into kernel category, Command set：

make ARCH=arm menuconfig

B. Device Drivers --->

Graphics support --->

Nexell Graphics --->

[ ] LVDS

[ ] MIPI

C. Select interface（LVDS 10 inch 1280*800）

[*] LVDS

rpdzkj lvds lcd select (RP LVDS LCD 1280*800 10 inch) --->

( ) RP LVDS LCD 1024*600 7 inch

( ) RP LVDS LCD 1024*600 10 inch

(X) RP LVDS LCD 1280*800 10 inch

Exit menuconfig and save compile kernel; select LVDS 10 inch 1280*800 LCD, then TP support resolution same as 1280*800.

3. Select LVDS, RP LVDS LCD 1280*800 10 inch, to CONFIG_NXP_DISPLAY_LVDS, CONFIG_LCD_LVDS_1280_800_10INCH set up value 1, at cfg_main.h, LCD parameter will be LVDS 1280*800 , references：

```
#if defined(CONFIG_NXP_DISPLAY_LVDS)
#if defined(CONFIG_LCD_LVDS_1024_600_10INCH)
#define CFG_DISP_PRI_RESOL_WIDTH                1024    // X Resolution
#define CFG_DISP_PRI_RESOL_HEIGHT               600 // Y Resolution

#define CFG_DISP_LVDS_LCD_FORMAT                LVDS_LCDFORMAT_JEIDA//LVDS_LCDFORMAT_VESA

#elif defined(CONFIG_LCD_LVDS_1024_600_7INCH)
#define CFG_DISP_PRI_RESOL_WIDTH                1024    // X Resolution
#define CFG_DISP_PRI_RESOL_HEIGHT               600 // Y Resolution

#define CFG_DISP_LVDS_LCD_FORMAT                LVDS_LCDFORMAT_VESA//LVDS_LCDFORMAT_JEIDA

#elif defined(CONFIG_LCD_LVDS_1280_800_10INCH)
#define CFG_DISP_PRI_RESOL_WIDTH                1280    // X Resolution
#define CFG_DISP_PRI_RESOL_HEIGHT               800 // Y Resolution

#define CFG_DISP_LVDS_LCD_FORMAT                LVDS_LCDFORMAT_JEIDA//LVDS_LCDFORMAT_VESA
```

4. TP resolution reference ft5x06_ts.h：

```
#if defined(CONFIG_LCD_LVDS_1024_600_10INCH) || defined(CONFIG_LCD_LVDS_1024_600_7INCH)
    #define SCREEN_MAX_X    1024    //for lvds 1024*600
    #define SCREEN_MAX_Y    600
#elif defined(CONFIG_LCD_LVDS_1280_800_10INCH)
    #define SCREEN_MAX_X    1280    //for lvds 1280*800
    #define SCREEN_MAX_Y    800
#elif defined(CONFIG_LCD_MIPI_1920_1200)
    #define SCREEN_MAX_X    1920    //for mipi 1920*1200
    #define SCREEN_MAX_Y    1200
#elif defined(CONFIG_LCD_RGB_800_480)
    #define SCREEN_MAX_X    800
    #define SCREEN_MAX_Y    480
#endif
```

Interface MIPI 10 inch 1920*1200 LCD, reference ft5x06_ts.c: ft5x0x_ts_report function

```
#if defined(CONFIG_LCD_MIPI_1920_1200)//for mipi 1920*1200
event->x[i] = event->x[i] * 1900 / 1280;
event->y[i] = event->y[i] * 1200 / 800;
#endif
```

5. If the parameter is not 1 on LCD,

example：1366*768,

then one have to modify：LCD

```
#define CFG_DISP_PRI_RESOL_WIDTH    1366// X Resolution
#define CFG_DISP_PRI_RESOL_HEIGHT   768 // Y Resolution

#define CFG_DISP_LVDS_LCD_FORMAT    LVDS_LCDFORMAT_VESA//根据实际修改，与uboot一致

#define CFG_DISP_PRI_CLKGEN0_DIV  18 // even divide
#define CFG_DISP_PRI_PIXEL_CLOCK  800000000/CFG_DISP_PRI_CLKGEN0_DIV
```

LCD Timing

```
#define SCREEN_MAX_X    1366
#define SCREEN_MAX_Y    768

event->x[i] = event->x[i] * 1366 / 1280;//beyond the max resolution 1280
event->y[i] = event->y[i] ;// stay in the max resolution 800
```

2. I2C Bus-Driving

1. I2C0 example：

```
static struct i2c_gpio_platform_data nxp_i2c_gpio_port0 = {
    .sda_pin    = I2C0_SDA,
    .scl_pin    = I2C0_SCL,
    .udelay     = I2CUDELAY(CFG_I2C0_CLK),
    /* Gpio_mode CLK Rate = 1/( udelay*2) * 1000000 */
    .timeout    = 10,
};

static struct platform_device i2c_device_ch0 = {
    .name   = "i2c-gpio",
    .id     = 0,
    .dev    = {
        .platform_data  = &nxp_i2c_gpio_port0,
    },
};
```

I2C0 is part of xp_i2c_gpio_port0 ：

.udelay = I2CUDELAY(CFG_I2C0_CLK)

```
if (pdata->udelay)
    bit_data->udelay = pdata->udelay;
else if (pdata->scl_is_output_only)
    bit_data->udelay = 50;          /* 10 kHz */
else
    bit_data->udelay = 5;           /* 100 kHz */
```

2. .udelay = I2CUDELAY(CFG_I2C0_CLK), decide I2C frequency is CFG_I2C0_CLK CFG_I2C0_CLK is at cfg_main.h decide.

#define CFG_I2C0_CLK 100000

3. Camera

Note：Camera module is changed,the driver is now ov5645.c. sp2518 code analysis is for reference only.

1. Camera driver code is sp2518.c

A. Camera is controlled through I2C, id_table is required, name is the same as driver module.

```
static const struct i2c_device_id sp2518_id[] = {
    { MODULE_NAME, 0 },
    { }
};
```

B. To opening the camera, the application will invoke HAL layer.
The V4L2 layer is used to open camera device, which calls the sp2518_init function eventually to reset the camera hardware,
writing specific values to the registers to make camera work properly.

```
GTP_GPIO_REQUEST(CAMERA_PD0,"CAMERA_PD0");
GTP_GPIO_REQUEST(CAMERA_RST,"CAMERA_RST");

GTP_GPIO_OUTPUT(CAMERA_PD0, 0);
mdelay(50);
GTP_GPIO_OUTPUT(CAMERA_RST, 1);
mdelay(50);//300

GTP_GPIO_OUTPUT(CAMERA_PD0, 1);
mdelay(50);//50
GTP_GPIO_OUTPUT(CAMERA_RST, 1);
mdelay(50);//50

GTP_GPIO_FREE(CAMERA_PD0);
GTP_GPIO_FREE(CAMERA_RST);
```

```
err = rp5065_i2c_write(sd, hm5065_init_reg1[i],
                       sizeof(hm5065_init_reg1[i]));
```

C. Camera is opened and taken a photo, it withh call sp2518_s_stream function to reset image format. Image format and resolution is determined by the values written into the registered.

```
err = rp5065_i2c_write(sd, hm5065_5M_reg[i],sizeof(hm5065_5M_reg[i]));
```

2. Camera parameters are in device.c

A. The camera control is through I2C driver.Registering the camera device on the I2C bus and implement the id_tabl is required. Therefore the i2c_board_info is required. The name is the same as the driver. The addres is the camera chip addres on I2C bus. It requires the the correct I2C address to obttain the data from the camera.

```
static struct i2c_board_info back_camera_i2c_boardinfo[] = {
    {
        I2C_BOARD_INFO("SP2518", 0x1f),
    },
};
```

B. The camera frame sync, pinout, clock rate and power control are all in device.c. All the controls ，are stored in the capture_plat_data[]
static struct nxp_capture_platformdata capture_plat_data[]

```
/* back_camera 656 interface */
.module = 0,// data channel 0, which is VIP0
.sensor = &sensor[0], // sensor[0]:2518; sensor[1]:0838
.type = NXP_CAPTURE_INF_PARALLEL,
.parallel = {
    /* for 656 */
    .is_mipi        = false,
    .external_sync  = true,
    .h_active       = 640,
    .h_frontporch   = 1,
    .h_syncwidth    = 1,
    .h_backporch    = 0,
    .v_active       = 480,
    .v_frontporch   = 0,
    .v_syncwidth    = 1,
    .v_backporch    = 0,
    .clock_invert   = false,
    .port           = 0,
    .data_order     =  NXP_VIN_CBY0CRY1,
    .interlace      = false,
    .clk_rate       = 24000000,
    .late_power_down = true,
    .power_enable   = back_camera_power_enable,
    .power_state_changed =back_camera_power_state_changed,
    .set_clock      = camera_common_set_clock,
    .setup_io       = camera_common_vin_setup_io,
},
.deci = {
    .start_delay_ms = 0,
    .stop_delay_ms  = 0,
},
```

3. HAL layer documents NXCameraHWInterface2.cpp, RP5065.cpp

A. When a camera applation is opened. It will invoke the framework layer then the HAL layer and eventually it will call the camera_device_open in the NXCameraHWInterface2.cpp. The camera device is activated at this function.

B. HAL layer source code directory

hardware\samsung_slsi\slsiap\camera
hardware\samsung_slsi\slsiap\v4l2
device\rpdzkj\rp6818\camera
framework layer source code
frameworks\av\camera

C. camera_device_open function NiXnC ameraHWInterface2.cpp

```
if (false == hal->init()) {
```
NXCameraHWInterface2.cpp

↓ call to

```
android_nxp_v4l2_init();
```
Android-nxp-v4l2.cpp

↓ call to

```
int ret = v4l2_init(&s);
```
Nxp-v4l2.cpp

↓ call to

```
int ret = _priv->init(scheme);
```

↓ call to

```
MediaFD = open("/dev/media0", O_RDWR);
if (MediaFD < 0) {
    ALOGE("can't open media device");
    return MediaFD;
}
```

eventually the device node is opened

D. RP5065.cpp setting up the image resolution.

```
if (width == 608)
    sensorWidth = 640;
else if(width == 2560)
 sensorWidth = 2592;
else if(width == 1024)
 sensorWidth = 1280;
else if(width == 1568 || width == 1560)
 sensorWidth = 1600;
 else {
    ALOGE("%s: invalid width %d", __func__, width);
  //  return -EINVAL;
    sensorWidth = width;
 }

 if (height == 479)
    sensorHeight = 480;
 else if(height == 1920)
    sensorHeight = 1944;
 else if (height == 1176 || height == 1170)
    sensorHeight = 1200;
 else if(height == 768)
    sensorHeight = 720;
 else {
    ALOGE("%s: invalid height %d", __func__, height);
 //    return -EINVAL;
    sensorHeight = height;
 }
```

4. GPIO pins and assignment

(Take J60 gpio as an example, refer to rp_gpio_ctrl.c)

1. function

J60 GPIO comes with 6 pinouts. 4 pins are available(J60-2～J60-5）, the other 2 is GND and 3.3V VSS. These 4 pins are general purpose and also can be assigned to other functions. Please refer to dataSheet of this SOC and cfg_gpio.h.

The reference schematic follows：

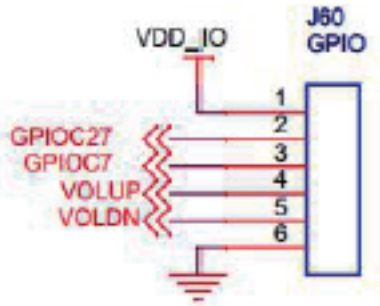


Fig 1

2. Pinout names and function assignment.

Befor operations on the pin out, one needs to figureout the pinout names in the driver. It is very important that if you got the wrong names,you will never be able to assign the functions and to access the pinout.

Take the J60-1 as an example to look up the pin out name

A. J60-1 has the name VOLUP in the schematic, shown in Fig 1.

B. VOLUP has the name GPIOB30 in the CPU page of the schematic,as shown in Fig2.

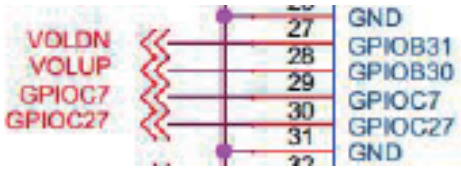


Fig 2

C. GPIOB30 is named in the driver as PAD_GPIO_B+30,then we can access to PAD_GPIO_B+30 to operate the pinout of VOLUP. Table 1 are the brief of the pingout's name and functions(please refer to arch/arm/plat-s5p6818/drone/include/cfg_gpio.h for more detail).

pinout	GPIO	name in the driver	general I/O function
0	GND		
1	GPIOB30	PAD_GPIO_B + 30	PAD_FUNC_ALT1
2	GPIOB31	PAD_GPIO_B + 31	PAD_FUNC_ALT1
3	GPIOC7	PAD_GPIO_C + 7	PAD_FUNC_ALT1
4	GPIOC27	PAD_GPIO_C + 27	PAD_FUNC_ALT1
5	3.3V		

Table 1

3. Function assignment(take J60-1 pinout as example)

It takes NX_GPIO_SetPadFunction(index, bit, func)to assign alternate function to the pinout.;parameter index to select the GPIO group：PAD_GPIO_B, parameter bit to select the GPIO pin：30, parameter func to select function：PAD_FUNC_ALT0/ PAD_FUNC_ALT1.

To assign the J60-2 as a general I/O：NX_GPIO_SetPadFunction(PAD_GPIO_B, 30, PAD_FUNC_ALT1)；

4. To assign these 4 pinout as general I/O,one can operate these pinouts as functional requirements.

Please refer to rp_gpio_ctrl.c
We also provide test application for Android plateform：RP_TEST.apk, this application can direct control these four pinout output level and display readback of the pinout voltage level.

5.Audio

1. wm8960.c and nxp-wm8960.c are the main files for audio control. Which are at kernel\sound\soc\codecs, nexell these two directory, one must make sure these two files are compiled in and have the probe invoked.

2. The audio relevant driver codes in device.c must be registered to match the driver, the probe can work only after that.

```
#if defined(CONFIG_SND_CODEC_WM8960)
#include <linux/i2c.h>
#define WM8976_I2C_BUS      (0)
/* CODEC */
static struct i2c_board_info __initdata wm8976_i2c_bdi = {
    .type   = "wm8960",          // compatilbe with wm8976
    .addr   = (0x34>>1),         // 0x1A (7BIT), 0x34(8BIT)
};
/* DAI */
struct nxp_snd_dai_plat_data i2s_dai_data = {
    .i2s_ch = 0,
    .sample_rate    = 48000,
    .pcm_format = SNDRV_PCM_FMTBIT_S16_LE,
    .hp_jack        = {
        .support        = 1,
        .detect_level   = 1,
    },
};

static struct platform_device wm8976_dai = {
    .name           = "wm8976-audio",
    .id             = 0,
    .dev            = {
        .platform_data  = &i2s_dai_data,
    }
};
#endif
```

3. The audio relevant options of the kernel compile in are required Under kernel directory make ARCH=arm menuconfig

Device Drivers --->
<*> Sound card support --->
<*> Advanced Linux Sound Architecture --->
<*> ALSA for SoC audio support --->
<*> (wm8960) I2S audio codec.

6.Key I/O

Note：Part of the code for key IO are altered, the operation of the driver and the sequence of the process are the same.

1. Key I/O device is plateform device, upon the key input driver is registered the system invoke the probe function and the driver will get the plateform device data：

in nxp_io_key.c struct nxp_key_plat_data * plat = pdev->dev.platform_data;plateform device data：in device.c

```
static unsigned int  button_gpio[] = CFG_KEYPAD_KEY_BUTTON;
static unsigned int  button_code[] = CFG_KEYPAD_KEY_CODE;

struct nxp_key_plat_data key_plat_data = {
    .bt_count   = ARRAY_SIZE(button_gpio),
    .bt_io      = button_gpio,
    .bt_code    = button_code,
    .bt_repeat  = CFG_KEYPAD_REPEAT,
};

static struct platform_device key_plat_device = {
    .name   = DEV_NAME_KEYPAD,
    .id     = -1,
    .dev    = {
        .platform_data  = &key_plat_data
    },
};
```

Correspond gpio for the Key and key definition are in cfg_main.h.

```
#define CFG_KEYPAD_KEY_BUTTON   { PAD_GPIO_D + 0,PAD_GPIO_C + 31,PAD_GPIO_B + 31, PAD_GPIO_B + 30, PAD_GPIO_ALV + 0 }
#define CFG_KEYPAD_KEY_CODE     { KEY_DOWN, KEY_UP, KEY_VOLUMEDOWN, KEY_VOLUMEUP, KEY_POWER }
```

One can realize the different functions for the by modifying the key code.
Yet, it still requires some modification in the interrupt service of the key.

2. Driver of the key invoked the input sub system, one has to define struct input_dev *input, allocating and initialize the memory space for it and then register the input device driver.

```
input = input_allocate_device();
if (NULL == input) {
    pr_err("fail, %s allocate input device\n", pdev->name);
    ret = -ENOMEM;
    goto err_mm;
}

 ret = input_register_device(input);
```

The event type will support input key type event and setup
the event type code for the input key type.

```
input->evbit[0] = BIT_MASK(EV_KEY);
```

Synchronizing the report events to make sure the system receive the event reported.

```
input_report_key(key->input, keycode, 1);
input_sync(key->input);
input_report_key(key->input, keycode, 0);
input_sync(key->input);
```

3. The android key referencing has to be done, only then the system can recognize the hardware key.

File location：
\lollipop_2nd_release\device\nexell\s5p6818_drone\keypad_s5p6818_drone.kl

key 114 VOLUME_DOWN WAKE
key 115 VOLUME_UP WAKE
key 116 POWER WAKE

One can Google it for the android key reference.

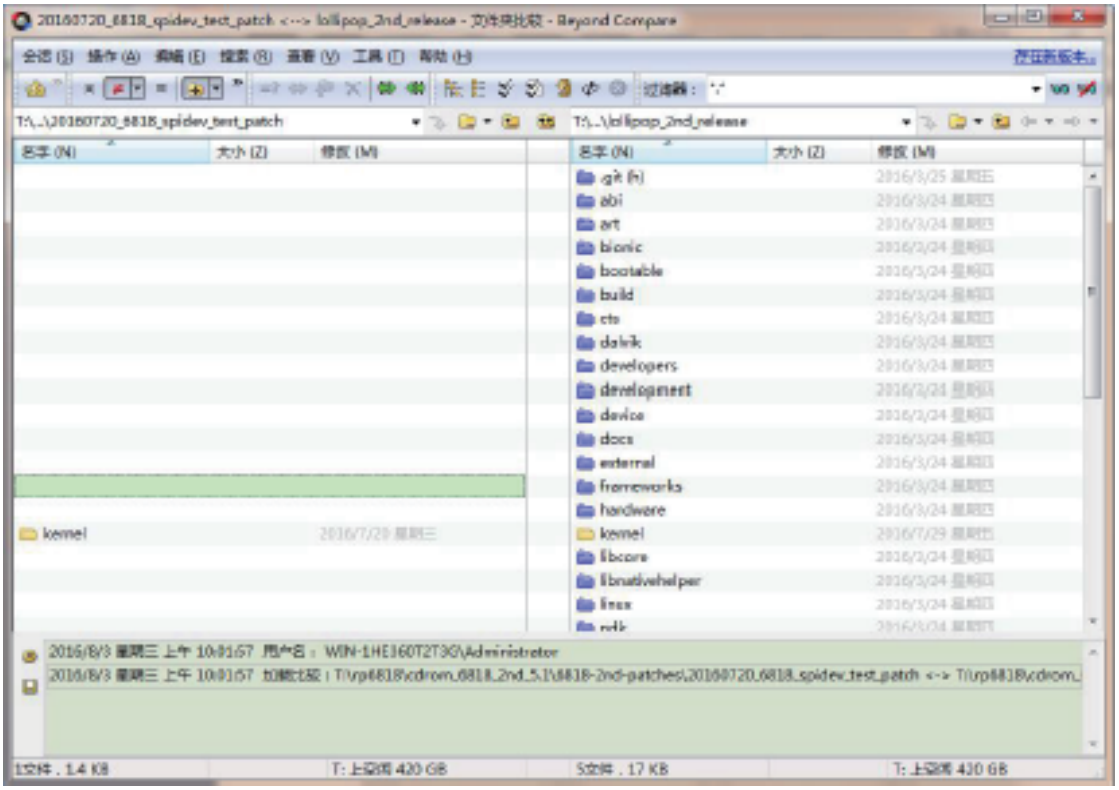
7.SPI

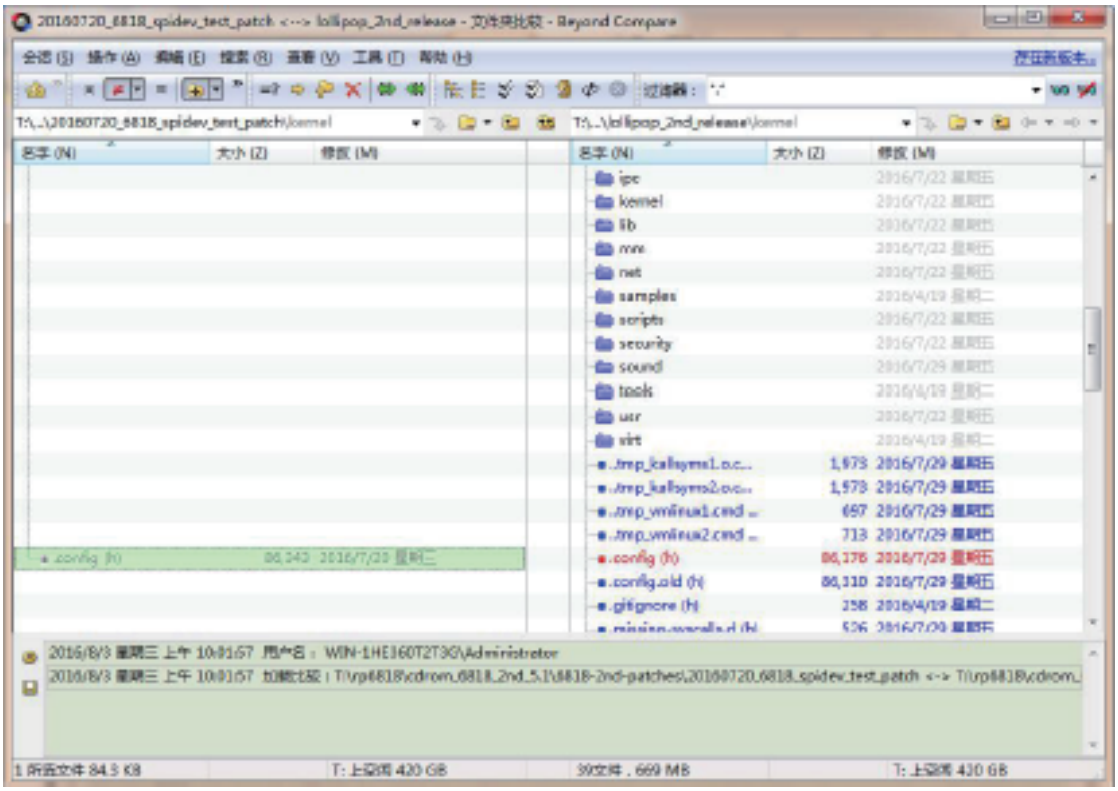
How to use spidev, how to run a C appliaction on a Android Linux kernel
(take 6818 as an example, 4418 is the same)

1. There is a SPIDev test patches for RP6818.

Directory: 20160720_6818_spidev_test_patch，this is a patch for tesing the SPI Bus and SPI deriver。
SPI driver module is spidev.c(kernel/drivers/spi).
Bus driver is spi-slsi.c(kernel/drivers/spi).
PI applications is under spi_test.

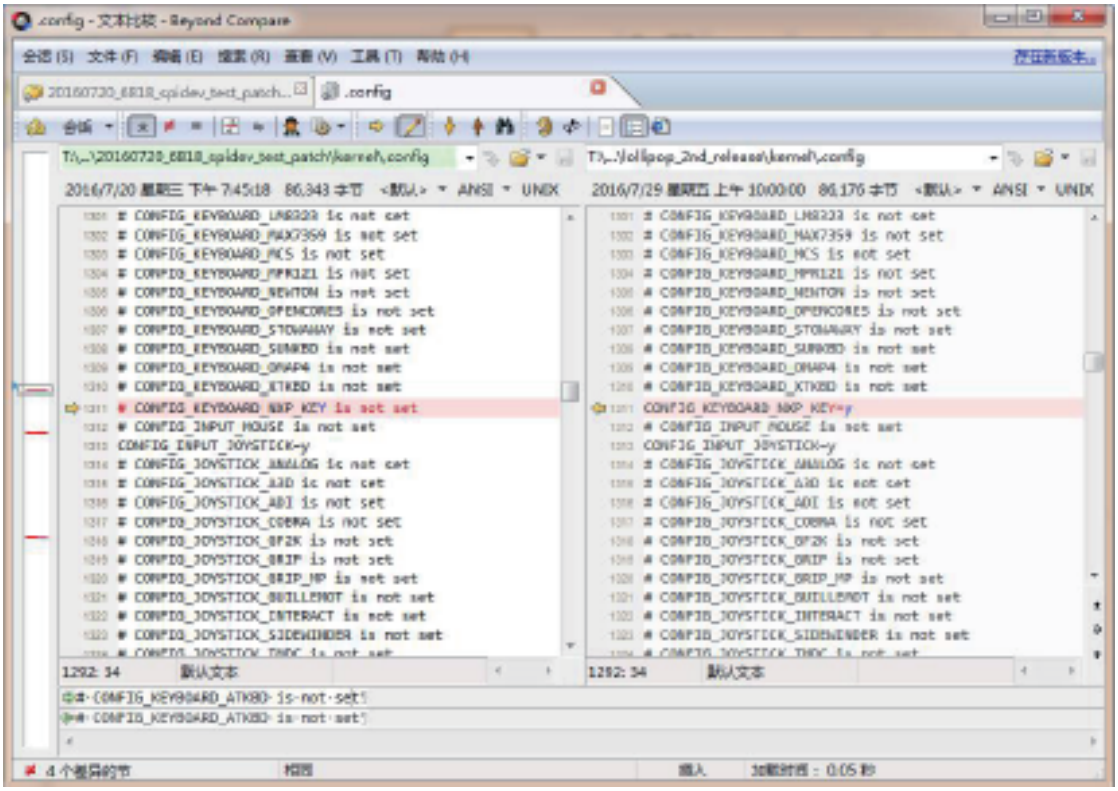
2. In the patch directoy, there is a hiden file. Config under directory kernel.
In which there are kernel compile options including the SPI driver and BUS.
The Beyond Compare can easily find the differences for you, then you would know what should be added to your kernel cofiguration.





3. Applying the patch to your source tree. Setting up the kernel optionsone has to alter the SPI relevant options only. Mainly two place of SPI relevant options are to be changed.

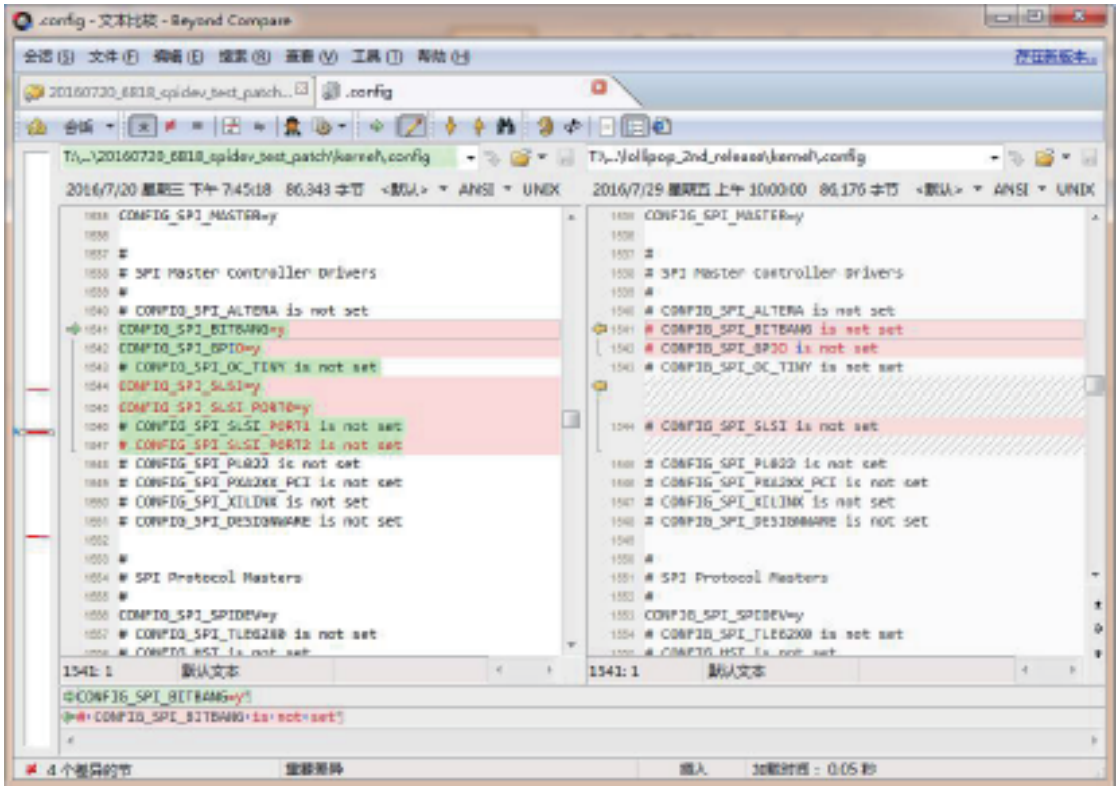
First option：



Remove the option of CONFIG_KEYBOARD_NXP_KEY.Because the SPI we are going to use is SPI0 which share the same pin whith GPIO.

4. Our pinout of the SPI0 appear at the EVB's J60 close to the RST key switch.CONFIG_KEYBOARD_NXP_KEY option's correspond driver will invoke these four pinouts one has to remove it temporarily.

Second option：



These options is for SPI BUS and SPIDev drivers which are all necessary.

The third places is relevant to LCD display configure it with your display option will be fine.

Eventually there is one important note. Alternating the .config and save it might not work.Because .config is automatically generated by kernel config.

To make the altered .config works, one should do this procedure once.Under the kernel directory run this comman :make ARCH=arm menuconfig. Entering the configuration menu then quit and save without alternating any thing.

Otherwise one can also follow the standard procedure to make it right.

Use the commane make ARCH=arm menuconfig to enter the menuconfig, find make the correct option settings and save it.

Except .config file, there are some other files need to be patched.Patch them according to the documents。Recompile your kernel after all files are patched.

5. Directory spi_test contains C applications for testing the SPI interface.There are two files, the Android.mk(can be edited) and spidev_test.c(source code of spidev_test in C language).

Android.mk contents：

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

LOCAL_SRC_FILES := spidev_test.c //file name and path, usually at the same directory as Android.mkded, the path is not required

LOCAL_C_INCLUDES := spidev_test.h //the header files if any, here we don't have any it is sudo name.

LOCAL_MODULE := spidev_test  //the name of executable after compiled.

LOCAL_MODULE_TAGS := optional

include $(BUILD_EXECUTABLE) //BUILD_EXECUTABLE to build executable, used other option to build a library, look up by youself
```

To build the Linux applications we use Makefile. To build Android application on Linux we should use Android.mk then

Build howtos:

A. Copy entire directory of spi_test to the directory of lollipop_2nd_release(which is also the directory of the android source tree)
B. ./build/envsetup.sh //script for setting up environment variables.
C. lunch //a production opetion list appears, we choose 13(which is 6818)
D. mmm spi_test //use the mmm comman to build, only after b,c are all done.
mmm command:
The generated spidev_test is located at out/target/product/s5p6818_drone/system/bin/
(there shold be printed message to show the location of the file).

6. Download the kernel image to the development board. Only the boot.img will be fine. Then reboot your device.

One can use the adb push to send the spidev_test to the development board(any way to send it is fine as long as the spidev_test can reach to the board), make the file spidev_test executable before the test.
So far the software preparation are all done. Next, you should shorten the two pins SPITXD0、SPIRXD0 of SPI0, the SPIRXD0 will receive the SPITXD0 data.

run it: ./spidev_test spi mode: 0
bits per word: 8
max speed: 498502 Hz (498 KHz)

FF FF FF FF FF FF
40 00 00 00 00 95
FF FF FF FF FF FF
FF FF FF FF FF FF
FF FF FF FF FF FF
DE AD BE EF BA AD
F0 0D

If the above message printed the bus spi0 and driver spidev works normally

8.Change the LOGO

The first boot up LOGO is currently the logo of we Winsonic. Follow the procedures to change the logo you want

Android:

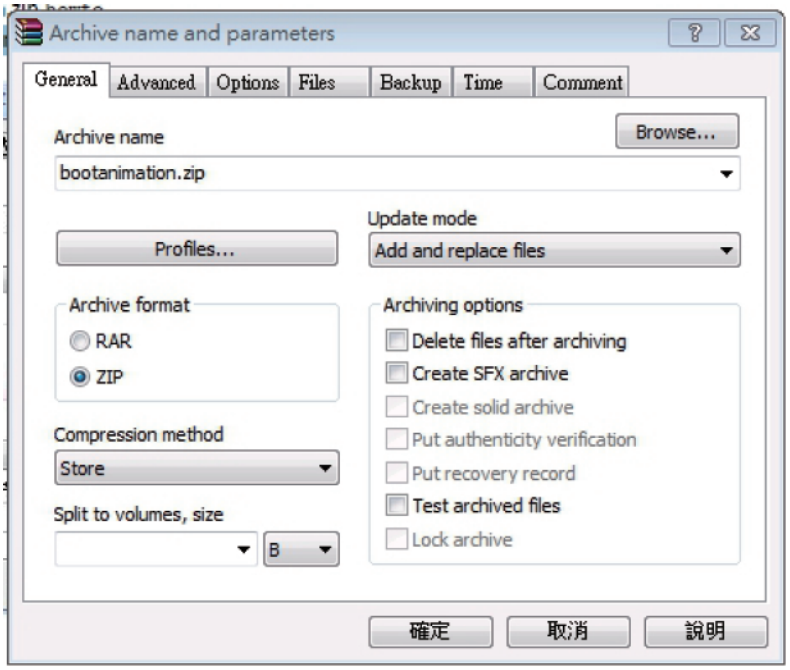
A. Build your own LOGO file in BMP format, the resolution of it better be the same as the LCD panel.
B. Rename your logo file to be logo.bmp. Replce the logo file with the same name at lollipop_2nd_release\device\nexell\s5p6818_drone\boot.
Note：for 4418, the directory is lollipop_2nd_release\device\nexell\s5p4418_drone\boot
C. Re-build the android，a new file of boot.img will be generated.

QT & Ubuntu:

A. Build your own LOGO file in BMP format, the resolution of it better be the same as the LCD panel.
B. Rename your logo file to be logo.bmp. Replce the logo file with the same name at QT_6818\result\boot.
Note:for 4418, the directory is QT_4418\result\boot
For Ubuntu, the path is at result\boot under the ubuntu directory.
C. Re-build the kernel，a new file of boot.img will be generated.

9. Change the Animation(Android)

Put your animation that is bootanimation.zip to system/media under the Android system.
So far we do not have the animation changed, the system shows the default original android one.
A. bootanimation.zip howto.
Google "bootanimation.zip" to find the way to.



The way to build is not introduced here.

B. Put bootanimation.zip to the source tree under the directory of device
6818：lollipop_2nd_release\device\nexell\s5p6818_drone
4418：lollipop_2nd_release\device\nexell\s5p4418_drone

C. One can edit the device.mk file.Add the following lines at a proper place.
PRODUCT_COPY_FILES += \
device/nexell/s5p6818_drone/bootanimation.zip:system/media/bootanimation.zip

```
PRODUCT_COPY_FILES += \
    device/nexell/s5p6818_drone/bootanimation.zip:system/media/bootanimation.zip
```

Note：4418 version will have different path

D. Re-build the android and a newer system.img will be generated.
Dwonload it to the board you will see the animation during the boot time.
NOTE ：If it shows that the animation won't be able to displayed,check the existence of your bootanimation.zip first. Seek it at lollipop_2nd_release\result\system\media. If it exists, then your animation build was not successful.