nanoPC-T2液晶驱动萃取

周星宇

2018.3.22

## 1.LCD设备

**1.0 总览**

在系统开机启动LCD初始化过程中，需要经历三个过程：

1 LCD型号选择

2 初始化LCD设备

3 初始化frameBuffer设备



**1.1 LCD型号**

NanoPi2支持的所有LCD型号使用一个结构体来存储nanopi2\_lcd\_config

1. /\* Try to guess LCD panel by kernel command line, or
2. \* using \*HD101\* as default
3. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
4. \*/
5. **static** **struct** {
6. **char** \*name;
7. **struct** nxp\_lcd \*lcd;
8. **int** ctp;
9. } nanopi2\_lcd\_config[] = {
10. { "HD101",  &wxga\_hd101, 1 },
11. { "HD101B", &wxga\_hd101, CTP\_GOODIX  },
12. { "HD700",  &wxga\_hd700, 1 },
13. { "HD702",  &wxga\_hd700, CTP\_GOODIX  },
14. { "S70",    &wvga\_s70,   1 },
15. { "S702",   &wvga\_s702,  1 },
16. { "S70D",   &wvga\_s70d,  0 },
17. { "X710",   &wsvga\_x710, CTP\_ITE7260 },
18. { "S430",   &wvga\_s430,  CTP\_HIMAX   },
19. #ifndef CONFIG\_ANDROID
20. { "H43",    &hvga\_h43,   0 },
21. { "P43",    &hvga\_p43,   0 },
22. { "W35",    &qvga\_w35,   0 },
23. #endif
24. /\* TODO: Testing \*/
25. { "W50",    &wvga\_w50,   0 },
26. { "W101",   &wsvga\_w101, 1 },
27. { "A97",    &xga\_a97,    0 },
28. { "LQ150",  &xga\_lq150,  1 },
29. { "L80",    &vga\_l80,    1 },
30. { "BP101",  &wxga\_bp101, 1 },
31. { "HDMI",   &hdmi\_def,   0 },   /\* Pls keep it at last \*/
32. };

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

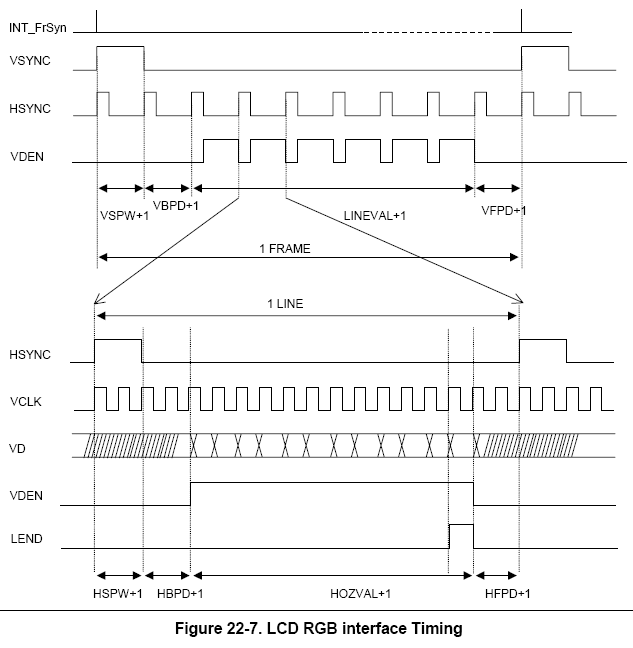
下面用wsvga\_x710型号的LCD举例。

1. /\*
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
3. \*/
4. **static** **struct** nxp\_lcd wsvga\_x710 = {
5. .width= 1024,
6. .height = 600,
7. .p\_width = 154,
8. .p\_height = 90,
9. .bpp = 24,
10. .freq = 61,
11. .timing = {
12. .h\_fp = 84,
13. .h\_bp = 84,
14. .h\_sw = 88,
15. .v\_fp = 10,
16. .v\_fpe = 1,
17. .v\_bp = 10,
18. .v\_bpe = 1,
19. .v\_sw = 20,
20. },
21. .polarity = {
22. .rise\_vclk = 0,
23. .inv\_hsync = 1,
24. .inv\_vsync = 1,
25. .inv\_vden = 0,
26. },
27. .gpio\_init = hd101\_gpio\_init,
28. };

其中各参数的含义如下：

1. /\*
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\include
3. \* struct nxp\_lcd
4. \* @width:      horizontal resolution <水平分辨率>
5. \* @height:     vertical resolution <垂直分辨率>
6. \* @p\_width:    width of lcd in mm <水平物理尺寸>
7. \* @p\_height:   height of lcd in mm <垂直物理尺寸>
8. \* @bpp:        bits per pixel <每个像素要用多少位(24)>
9. \* @freq:       vframe frequency <刷新帧率(61)>
10. \* @timing:     timing values <存储时序参数的结构体>
11. \* @polarity:   polarity settings <结构体，存储信号线极性>
12. \* @gpio\_init:  pointer to GPIO init function <GPIO初始化函数>
13. \*
14. \*/
15. **struct** nxp\_lcd {
16. **int** width;
17. **int** height;
18. **int** p\_width;
19. **int** p\_height;
20. **int** bpp;
21. **int** freq;
22. **struct**  nxp\_lcd\_timing timing;
23. **struct**  nxp\_lcd\_polarity polarity;
24. **void**    (\*gpio\_init)(**void**);
25. };
26. /\*
27. \* struct nxp\_lcd\_timing
28. \* @h\_fp:   horizontal front porch
29. \* @h\_bp:   horizontal back porch
30. \* @h\_sw:   horizontal sync width
31. \* @v\_fp:   vertical front porch
32. \* @v\_fpe:  vertical front porch for even field
33. \* @v\_bp:   vertical back porch
34. \* @v\_bpe:  vertical back porch for even field
35. \*/
36. **struct** nxp\_lcd\_timing {
37. **int** h\_fp;
38. **int** h\_bp;
39. **int** h\_sw;
40. **int** v\_fp;
41. **int** v\_fpe;
42. **int** v\_bp;
43. **int** v\_bpe;
44. **int** v\_sw;
45. };

时序参数的含义参考LCD控制时序



1. /\*
2. \* struct nxp\_lcd\_polarity
3. \* @rise\_vclk:  if 1, video data is fetched at rising edge
4. \* @inv\_hsync:  if HSYNC polarity is inversed
5. \* @inv\_vsync:  if VSYNC polarity is inversed
6. \* @inv\_vden:   if VDEN polarity is inversed
7. \*/
8. **struct** nxp\_lcd\_polarity {
9. **int** rise\_vclk;
10. **int** inv\_hsync;
11. **int** inv\_vsync;
12. **int** inv\_vden;
13. };

这些LCD参数在系统的配置过程中有两个去向，分别用来初始化两个platform设备：

1 LCD 的vsync参数->LCD设备

2 Framebuffer参数->framebuffer设备

**1.2 lcd设备初始化**

LCD设备使用platform设备驱动框架，由一个platform\_device结构体来描述。

该设备对应的驱动程序用来控制s5p4418芯片中LCD控制器，从而实现对LCD液晶屏显示的控制。

1. //linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\mach-s5p4418\dev-display.c
2. **static** **struct** disp\_lcd\_param   \_\_lcd\_devpar;
4. **static** **struct** nxp\_lcd\_plat\_data lcd\_data = {
5. .display\_in     = DISPLAY\_INPUT(CONFIG\_NXP\_DISPLAY\_LCD\_IN),
6. .display\_dev    = DISP\_DEVICE\_LCD,
7. .vsync          = &\_\_lcd\_vsync,
8. .dev\_param      = (**union** disp\_dev\_param\*)&\_\_lcd\_devpar,
9. };
10. //LCD设备
11. **static** **struct** platform\_device lcd\_device = {
12. .name   = DEV\_NAME\_LCD,
13. .id     = -1,
14. .dev    = {
15. .platform\_data  = &lcd\_data
16. },
17. };

其中lcd\_data中的vsync成员中存储了LCD控制时序相关的参数

1. **static** **struct** disp\_vsync\_info \_\_lcd\_vsync = {
2. /\* default parameters refer to cfg\_main.h \*/
3. #if defined(CFG\_DISP\_PRI\_RESOL\_WIDTH) && defined(CFG\_DISP\_PRI\_RESOL\_HEIGHT)
4. .h\_active\_len   = CFG\_DISP\_PRI\_RESOL\_WIDTH,
5. .h\_sync\_width   = CFG\_DISP\_PRI\_HSYNC\_SYNC\_WIDTH,
6. .h\_back\_porch   = CFG\_DISP\_PRI\_HSYNC\_BACK\_PORCH,
7. .h\_front\_porch  = CFG\_DISP\_PRI\_HSYNC\_FRONT\_PORCH,
8. .h\_sync\_invert  = CFG\_DISP\_PRI\_HSYNC\_ACTIVE\_HIGH,
9. .v\_active\_len   = CFG\_DISP\_PRI\_RESOL\_HEIGHT,
10. .v\_sync\_width   = CFG\_DISP\_PRI\_VSYNC\_SYNC\_WIDTH,
11. .v\_back\_porch   = CFG\_DISP\_PRI\_VSYNC\_BACK\_PORCH,
12. .v\_front\_porch  = CFG\_DISP\_PRI\_VSYNC\_FRONT\_PORCH,
13. .v\_sync\_invert  = CFG\_DISP\_PRI\_VSYNC\_ACTIVE\_HIGH,
14. .pixel\_clock\_hz = CFG\_DISP\_PRI\_PIXEL\_CLOCK,
15. .clk\_src\_lv0    = CFG\_DISP\_PRI\_CLKGEN0\_SOURCE,
16. .clk\_div\_lv0    = CFG\_DISP\_PRI\_CLKGEN0\_DIV,
17. .clk\_src\_lv1    = CFG\_DISP\_PRI\_CLKGEN1\_SOURCE,
18. .clk\_div\_lv1    = CFG\_DISP\_PRI\_CLKGEN1\_DIV,
19. #endif
20. };

其中的参数初始值的宏定义在板子相关的初始化文件cfg\_main.h中：

1. // \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\include\cfg\_main.h
2. #define CFG\_DISP\_PRI\_RESOL\_WIDTH                1280    // X Resolution
3. #define CFG\_DISP\_PRI\_RESOL\_HEIGHT               800 // Y Resolution
5. #define CFG\_DISP\_PRI\_HSYNC\_SYNC\_WIDTH           30
6. #define CFG\_DISP\_PRI\_HSYNC\_BACK\_PORCH           16
7. #define CFG\_DISP\_PRI\_HSYNC\_FRONT\_PORCH          16
8. #define CFG\_DISP\_PRI\_HSYNC\_ACTIVE\_HIGH          CTRUE
9. #define CFG\_DISP\_PRI\_VSYNC\_SYNC\_WIDTH           12
10. #define CFG\_DISP\_PRI\_VSYNC\_BACK\_PORCH           8
11. #define CFG\_DISP\_PRI\_VSYNC\_FRONT\_PORCH          8
12. #define CFG\_DISP\_PRI\_VSYNC\_ACTIVE\_HIGH          CTRUE

但是加载驱动时的参数值并不再是这些宏定义中的初始值，

nxp\_platform\_disp\_init函数将nanopi2\_lcd\_config[]中对应型号的LCD参数赋值到\_\_lcd\_vsync中。实现LCD设备参数和1.1节描述的参数同一性。

**2.3 frameBuffer设备**

1. /\*------------------------------------------------------------------------------
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
3. \*/
4. **static** **struct** platform\_device \*fb\_devices[] = {
5. #if defined (CONFIG\_FB0\_NXP)
6. &fb0\_device,
7. #endif
8. };
10. /\*
11. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
12. \*/
13. #if defined (CONFIG\_FB0\_NXP)
14. **static** **struct** platform\_device fb0\_device = {
15. .name   = DEV\_NAME\_FB,
16. .id     = 0,    /\* FB device node num \*/
17. .dev    = {
18. .coherent\_dma\_mask  = 0xffffffffUL, /\* for DMA allocate \*/
19. .platform\_data      = &fb0\_plat\_data
20. },
21. };
23. **static** **struct** nxp\_fb\_plat\_data fb0\_plat\_data = {
24. .module         = CONFIG\_FB0\_NXP\_DISPOUT,
25. .layer          = CFG\_DISP\_PRI\_SCREEN\_LAYER,
26. #ifdef CONFIG\_FB\_NXP\_X8R8G8B8
27. .format         = MLC\_RGBFMT\_X8R8G8B8,
28. #else
29. .format         = CFG\_DISP\_PRI\_SCREEN\_RGB\_FORMAT,
30. #endif
31. .bgcolor        = CFG\_DISP\_PRI\_BACK\_GROUND\_COLOR,
32. .bitperpixel    = CFG\_DISP\_PRI\_SCREEN\_PIXEL\_BYTE \* 8,
33. .x\_resol        = CFG\_DISP\_PRI\_RESOL\_WIDTH,
34. .y\_resol        = CFG\_DISP\_PRI\_RESOL\_HEIGHT,
35. #ifdef CONFIG\_ANDROID
36. .buffers        = 3,
37. .skip\_pan\_vsync = 1,
38. #else
39. .buffers        = 2,
40. #endif
41. .lcd\_with\_mm    = CFG\_DISP\_PRI\_LCD\_WIDTH\_MM,    /\* 152.4 \*/
42. .lcd\_height\_mm  = CFG\_DISP\_PRI\_LCD\_HEIGHT\_MM,   /\* 91.44 \*/
43. };
44. #endif

其中默认值定义如下：

1. /\*
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\include
3. \* fb0\_plat\_data结构中的部分信息
4. \*/
5. #define CFG\_DISP\_PRI\_LCD\_WIDTH\_MM               218
6. #define CFG\_DISP\_PRI\_LCD\_HEIGHT\_MM              136
7. #define CFG\_DISP\_PRI\_RESOL\_WIDTH                1280   // X Resolution
8. #define CFG\_DISP\_PRI\_RESOL\_HEIGHT               800 // Y Resolution

该结构也需要在初始化时期载入1.1描述的对应型号LCD参数。

**1.4 驱动加载流程**

在linux内核启动之前，uboot已经进行了LCD型号识别的工作。

uboot使用1wire通信协议和LCD控制板进行交互，读取LCD型号的信息，并将该信息以linux内核启动参数的形式传递给linux内核。

从而linux内核能在众多支持的LCD型号中选择出使用的型号。



本节描述LCD驱动加载流程中调用的各函数

1. /\*------------------------------------------------------------------------------
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
3. \*/
4. #if defined(CONFIG\_NXP\_DISPLAY)
5. **static** **void** nxp\_platform\_disp\_init(**struct** nxp\_lcd \*lcd){
6. **struct** disp\_vsync\_info vsync;
7. **struct** nxp\_lcd\_timing \*timing;
8. u32 clk = 800000000;
9. u32 div;
11. **if** (lcd) {
12. timing = &lcd->timing;
14. vsync.interlace     = 0;
15. vsync.h\_active\_len  = lcd->width;
16. vsync.h\_sync\_width  = timing->h\_sw;
17. vsync.h\_back\_porch  = timing->h\_bp;
18. vsync.h\_front\_porch = timing->h\_fp;
19. vsync.h\_sync\_invert = !lcd->polarity.inv\_hsync;
21. vsync.v\_active\_len  = lcd->height;
22. vsync.v\_sync\_width  = timing->v\_sw;
23. vsync.v\_back\_porch  = timing->v\_bp;
24. vsync.v\_front\_porch = timing->v\_fp;
25. vsync.v\_sync\_invert = !lcd->polarity.inv\_vsync;
27. /\* calculates pixel clock \*/
28. div  = timing->h\_sw + timing->h\_bp + timing->h\_fp + lcd->width;
29. div \*= timing->v\_sw + timing->v\_bp + timing->v\_fp + lcd->height;
30. div \*= lcd->freq ? : 60;
31. do\_div(clk, div);
33. vsync.pixel\_clock\_hz= div;
34. vsync.clock\_gen\_num = 0;
35. vsync.clk\_src\_lv0   = CFG\_DISP\_PRI\_CLKGEN0\_SOURCE;
36. vsync.clk\_div\_lv0   = clk;
37. vsync.clk\_src\_lv1   = CFG\_DISP\_PRI\_CLKGEN1\_SOURCE;
38. vsync.clk\_div\_lv1   = CFG\_DISP\_PRI\_CLKGEN1\_DIV;
39. vsync.clk\_out\_inv   = lcd->polarity.rise\_vclk;
41. **if** (lcd->gpio\_init)
42. lcd->gpio\_init();
44. #if defined(CONFIG\_NXP\_DISPLAY\_LCD)
45. nxp\_platform\_disp\_device\_data(DISP\_DEVICE\_LCD,  &vsync, NULL, NULL);
46. #endif
47. #if defined(CONFIG\_NXP\_DISPLAY\_LVDS)
48. nxp\_platform\_disp\_device\_data(DISP\_DEVICE\_LVDS, &vsync, NULL, NULL);
49. #endif
50. }
51. }
52. #endif
53. /\*
54. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
55. \*/
56. **static** **void** nxp\_platform\_fb\_data(**struct** nxp\_lcd \*lcd){
57. #if defined (CONFIG\_FB0\_NXP)
58. **struct** nxp\_fb\_plat\_data \*pdata = &fb0\_plat\_data;
60. **if** (lcd) {
61. pdata->x\_resol = lcd->width;
62. pdata->y\_resol = lcd->height;
64. #if defined (CONFIG\_FB0\_NXP\_FIXED\_DPI)
65. pdata->lcd\_with\_mm = (lcd->width \* 254) / 1310;
66. pdata->lcd\_height\_mm = (lcd->height \* 254) / 1310;
67. #else
68. pdata->lcd\_with\_mm = lcd->p\_width;
69. pdata->lcd\_height\_mm = lcd->p\_height;
70. #endif
71. }
72. #endif
73. }
74. /\*\*
75. \* \linux-3.4.y-nanopi2-lollipop-mr1\drivers\base
76. \* platform\_add\_devices - add a numbers of platform devices
77. \* @devs: array of platform devices to add
78. \* @num: number of platform devices in array
79. \*/
80. **int** platform\_add\_devices(**struct** platform\_device \*\*devs, **int** num){
81. **int** i, ret = 0;
82. **for** (i = 0; i < num; i++) {
83. ret = platform\_device\_register(devs[i]);
84. **if** (ret) {
85. **while** (--i >= 0)
86. platform\_device\_unregister(devs[i]);
87. **break**;
88. }
89. }
90. **return** ret;
91. }

## 3.触摸

**3.1 编译驱动**

首先linux内核要支持对应的触摸芯片驱动。

修改.config文件，添加触摸屏驱动支持

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 is not set
5. # CONFIG\_TOUCHSCREEN\_ST1232 is not set
6. # CONFIG\_TOUCHSCREEN\_TPS6507X is not set
7. CONFIG\_TOUCHSCREEN\_FT5X0X=y

**3.2 定义I2c设备**

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. #if defined(CONFIG\_TOUCHSCREEN\_IT7260)
2. #define IT7260\_I2C\_BUS      (2)
4. **static** **struct** i2c\_board\_info \_\_initdata it7260\_i2c\_bdi = {
5. I2C\_BOARD\_INFO("IT7260", (0x8C>>1)),
6. .irq = PB\_PIO\_IRQ(CFG\_IO\_TOUCH\_IRQ),
7. };
8. #endif

其中需要注意：

1.I2C设备的地址怎么写(例子中为0x8C>>1)

2."IT7260"为I2C设备名称，需要和驱动IT7260设备驱动程序中的驱动名称一致。

驱动程序定义如下所示：

1. //linux-3.4.y-nanopi2-lollipop-mr1\drivers\input\touchscreen\it7260\_mts.c
2. **static** **const** **struct** i2c\_device\_id it7260\_ts\_id[] = {
3. {"IT7260", 0},
4. {}          /\* should not omitted \*/
5. };
6. MODULE\_DEVICE\_TABLE(i2c, it7260\_ts\_id);
8. **static** **struct** i2c\_driver it7260\_ts\_driver = {
9. .driver = {
10. .name = "IT7260-ts",
11. },
12. .probe = it7260\_ts\_probe,
13. .remove = \_\_devexit\_p(it7260\_ts\_remove),
14. .suspend = it7260\_ts\_suspend,
15. .resume = it7260\_ts\_resume,
16. .id\_table = it7260\_ts\_id,
17. };

**3.3 触摸屏设备加载**

加载总体流程如下图所示：



在开机初始化函数nxp\_board\_devices\_register(**void**)中加载3.2中定义的触摸触摸屏设备。

注册的新触摸驱动需要在该函数中进行加载

1. //linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\device.c
2. **void** \_\_init nxp\_board\_devices\_register(**void**){
3. //.................
4. #if defined(CONFIG\_TOUCHSCREEN\_FT5X0X)
5. printk("plat: add touch(ft5x0x) device\n");
6. ft5x0x\_pdata.screen\_max\_x = lcd->width;
7. ft5x0x\_pdata.screen\_max\_y = lcd->height;
8. i2c\_register\_board\_info(FT5X0X\_I2C\_BUS, &ft5x0x\_i2c\_bdi, 1);
9. #endif
11. #if defined(CONFIG\_TOUCHSCREEN\_GOODIX)
12. printk("plat: add touch(goodix) device\n");
13. goodix\_pdata.screen\_max\_x = lcd->width;
14. goodix\_pdata.screen\_max\_y = lcd->height;
15. i2c\_register\_board\_info(GOODIX\_I2C\_BUS, &goodix\_i2c\_bdi, 1);
16. printk("plat: goodix: irq=%d (%d)\n", PB\_PIO\_IRQ(CFG\_IO\_TOUCH\_IRQ), CFG\_IO\_TOUCH\_IRQ);
17. #endif
19. #if defined(CONFIG\_TOUCHSCREEN\_HIMAX)
20. printk("plat: add touch(himax) device\n");
21. i2c\_register\_board\_info(HIMAX\_I2C\_BUS, &himax\_i2c\_bdi, 1);
22. #endif
24. #if defined(CONFIG\_TOUCHSCREEN\_IT7260)
25. printk("plat: add touch(it7260) device\n");
26. i2c\_register\_board\_info(IT7260\_I2C\_BUS, &it7260\_i2c\_bdi, 1);
27. #endif
28. //.......................

如果系统中已经内置了对应型号的驱动，且驱动名称和设备名称一致。Probe函数就会被调用。Probe函数中进行一些工作：

注册input子系统

注册中断处理函数。。。。等等。

如下图所示：



**3.4 触摸驱动型号确认**

触摸芯片型号和LCD型号相关联

1. //linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\lcds.c
2. **static** **int** \_\_init nanopi2\_init\_ctp(**char** \*str)
3. {
4. unsigned **int** val;
5. **char** \*p = str, \*end;
7. val = simple\_strtoul(p, &end, 10);
8. **if** (end <= p) {
9. **return** 1;
10. }
12. **if** (val < CTP\_MAX && nanopi2\_lcd\_config[lcd\_idx].ctp) {
13. ctp\_type = val;
14. } **else** **if** (val == CTP\_NONE) {
15. ctp\_type = CTP\_NONE;
16. }
18. **return** 1;
19. }
20. \_\_setup("ctp=", nanopi2\_init\_ctp);
22. unsigned **int** nanopi2\_get\_ctp(**void**)
23. {
24. **if** (nanopi2\_lcd\_config[lcd\_idx].ctp)
25. **return** ctp\_type;
26. **else**
27. **return** CTP\_NONE;
28. }
29. EXPORT\_SYMBOL(nanopi2\_get\_ctp);
31. **void** nanopi2\_set\_ctp(**int** type)
32. {
33. **if** (ctp\_type == CTP\_AUTO && type < CTP\_MAX) {
34. ctp\_type = type;
35. }
36. }
37. EXPORT\_SYMBOL(nanopi2\_set\_ctp);
38. **static** **inline** **int** board\_get\_ctp(**void**) {
39. **return** nanopi2\_get\_ctp();
40. }

## 3.背光驱动



1. //linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2\device.c
2. #if defined(CONFIG\_BACKLIGHT\_PWM)
3. #include <linux/pwm\_backlight.h>
5. **static** **struct** platform\_pwm\_backlight\_data bl\_plat\_data = {
6. .pwm\_id         = CFG\_LCD\_PRI\_PWM\_CH,
7. .max\_brightness = 400,  /\* 255 is 100%, set over 100% \*/
8. .dft\_brightness = 128,  /\* 50% \*/
9. .pwm\_period\_ns  = 1000000000/CFG\_LCD\_PRI\_PWM\_FREQ,
10. };
12. **static** **struct** platform\_device bl\_plat\_device = {
13. .name   = "pwm-backlight",
14. .id     = -1,
15. .dev    = {
16. .platform\_data  = &bl\_plat\_data,
17. },
18. };
19. #endif

对应驱动程序如下

1. //linux-3.4.y-nanopi2-lollipop-mr1\drivers\video\backlight\pwm-bl.c
2. **static** **struct** platform\_driver pwm\_backlight\_driver = {
3. .driver     = {
4. .name   = "pwm-backlight",
5. .owner  = THIS\_MODULE,
6. #ifdef CONFIG\_PM
7. .pm = &pwm\_backlight\_pm\_ops,
8. #endif
9. },
10. .probe      = pwm\_backlight\_probe,
11. .remove     = pwm\_backlight\_remove,
12. };

## 4.系统启动时注册设备过程

系统启动时首先调用`\linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\mach-s5p4418`中的`cpu\_init\_machine`方法。`cpu\_init\_machine`中分别调用两个函数注册系统的各种设备。

\* `nxp\_cpu\_devices\_register()`函数

\* `nxp\_board\_devices\_register()`函数

1. /\*
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\mach-s5p4418
3. \*/
4. **static** **void** \_\_init cpu\_init\_machine(**void**){
5. /\* set shutdown \*/
6. pm\_power\_off   = nxp\_cpu\_shutdown;
7. arm\_pm\_restart = nxp\_cpu\_reset;
8. /\*
9. \* register platform device
10. \*/
11. nxp\_cpu\_devices\_register();  //1.1：
12. nxp\_board\_devices\_register(); //1.2：
13. }

1.1`nxp\_cpu\_devices\_register()`函数的实现如下所示。在其中根据系统宏定义（如CONFIG\_NXP\_DISPLAY\_LCD）来判断注册显示设备的类型，可选类型有LCD、LVDS、MIPI、HDMI。并使用`platform\_device\_register`函数将设备注册到系统中。

1. /\*------------------------------------------------------------------------------
2. \* register cpu platform devices
3. \*/
4. **void** \_\_init nxp\_cpu\_devices\_register(**void**){
6. ......
7. //通过检查系统是否定义宏CONFIG\_NXP\_DISPLAY\_LCD来判断是否使用LCD设备
8. #if defined(CONFIG\_NXP\_DISPLAY\_LCD)
9. printk("mach: add device lcd \n");
10. platform\_device\_register(&lcd\_device);   //1.3：注册LCD设备
11. #endif
12. #if defined(CONFIG\_NXP\_DISPLAY\_LVDS)
13. printk("mach: add device lvds \n");
14. platform\_device\_register(&lvds\_device);
15. #endif
16. #if defined(CONFIG\_NXP\_DISPLAY\_MIPI)
17. printk("mach: add device mipi \n");
18. platform\_device\_register(&mipi\_device);
19. #endif
20. #if defined(CONFIG\_NXP\_DISPLAY\_HDMI)
21. printk("mach: add device hdmi \n");
22. platform\_device\_register(&hdmi\_device);
23. #endif
25. ......
26. }

1.2`nxp\_board\_devices\_register()`函数的实现如下所示。

\* 使用`nanopi2\_get\_lcd()`函数获取系统使用的LCD设备型号。

\* 使用`nxp\_platform\_disp\_init(lcd)`函数

\* 使用`nxp\_platform\_fb\_data(lcd)`函数

\* 使用`platform\_add\_devices`函数

1. /\*------------------------------------------------------------------------------
2. \* \linux-3.4.y-nanopi2-lollipop-mr1\arch\arm\plat-s5p4418\nanopi2
3. \* register board platform devices
4. \*/
5. **void** \_\_init nxp\_board\_devices\_register(**void**){
6. **struct** nxp\_lcd \*lcd = nanopi2\_get\_lcd();    //1.4：获取系统使用的LCD设备型号
8. 。。。。。。
10. #if defined(CONFIG\_NXP\_DISPLAY)
11. nxp\_platform\_disp\_init(lcd);    //1.5
12. #endif
13. #if defined(CONFIG\_FB\_NXP)
14. printk("plat: add framebuffer\n");
15. nxp\_platform\_fb\_data(lcd);    //1.6
16. platform\_add\_devices(fb\_devices, ARRAY\_SIZE(fb\_devices));    //1.7
17. #endif
19. 。。。。。。
21. #if defined(CONFIG\_TOUCHSCREEN\_FT5X0X)   //根据液晶尺寸来配置触摸板参数
22. printk("plat: add touch(ft5x0x) device\n");
23. ft5x0x\_pdata.screen\_max\_x = lcd->width;
24. ft5x0x\_pdata.screen\_max\_y = lcd->height;
25. i2c\_register\_board\_info(FT5X0X\_I2C\_BUS, &ft5x0x\_i2c\_bdi, 1);
26. #endif
28. 。。。。。。
29. }